

Hitachi MB-6890 user and BASIC programming manual (ca. 1980)

The contents of this manual or any part thereof may not be copied without permission.

The content of this manual may be changed without notice.

CONTENTS

CHAPTER 1	:	LEVEL-3 - OUTLINE	Page 1
1.		Level-3 Capabilities and Key Functions	1
2.		Screen Mode Change	11
	*	Interlace mode and non-interlace mode	11
	*	Graphic mode	12
	*	Colour display	14
	*	Multipage	15
	*	Dip switch setting	16
3.		Connection to Peripherals	16
4.		Expandability of Level-3	17
CHAPTER 2	:	BASIC	19
		Fundamental Statements	19
	*	PRINT - Output to the screen	19
	*	Variables . Substitutes	25
	*	INPUT - Inputting data to the variables from the keyboard	27
	*	GOTO - Changing the program flow	29
	*	IF - Changing the program flow	29
	*	FOR-NEXT loop - Repeat jobs	32
	*	GOSUB-RETURN - Subroutines	35
	*	Summary of the basic statements and commands	37
	*	Important points to remember when you write the program	39
CHAPTER 3	:	POWER ON	41
	*	The Return key	42
	*	Screen editor	43
CHAPTER 4	:	LEVEL-3 BASIC	46
1.		Level-3 BASIC Fundamental Knowledge	46
	*	File Descriptor	46
	*	Constants . Variables	48
2.		Mastering the Commands	48
	*	LIST - Display the program in the memory	48

* RUN - Program execution	Page 51
* CONT - Re-start the program	51
* NEW - Erase the program	52
3. Making Programming Easier	52
* AUTO - Automatic line number generator	52
* RENUM - Renumber the line numbers	54
* DELETE - Delete a part of the program	55
* EDIT - Edit the program	56
* TRON, TROFF - To check the flow of the program	57
CHAPTER 5 : STATEMENTS	59
1. LET statement	59
2. Input/Output Instructions	59
* INPUT - Basic statement of input	60
* LINE INPUT - Any characters are input	61
* INPUT WAIT - Input with a time limit	62
* PRINT - Basic statement for output	63
* PRINT USING - Format control output	66
* ^^ ^^ - Exponent display designation	71
* Application of PRINT USING	72
* READ, DATA, RESTORE	74
3. Other Important Statements	76
* FOR-NEXT loop - Repeat executions	77
* DIM - Array variables	80
* IF-THEN-ELSE program	81
* ON-GOTO, ON-GOSUB - Transfer program control by calculation results	82
* DEF INT/SNG/DBL/STR	83
* User Function Definition	84
* SWAP - To exchange the values of two variables	86
4. Error Procedures	87
* ON ERROR GOTO, RESUME - Error-handling routine	87
* ERR, ERL - Error information	89
* ERROR - Error generation	89

CHAPTER 6 : MASTERING THE FUNCTIONS

Page 91

1. Numerical Functions	91
* SIN, COS, TAN, ATN - Trigonometrical functions	91
* SQR - Square root	94
* EXP - Exponential function	95
* LOG - Natural logarithms	95
* Common logarithms	96
* ABS, SGN - Absolute value	96
* INT, FIX, CINT - To make integers	97
* CSNG, CDBL - Conversion function	98
* RND - Random numbers	99
2. Character Functions	100
* LEFT\$, RIGHT\$, MID\$ - Handling character strings	100
* LEN - To check the length of the character string	102
* ASC, CHR\$ - Code conversion	102
* STR\$, VAL - Numeric character conversion	103
* SPACE\$, STRING\$ - Character generator function	104
* INSTR - Searching function	104
* INKEY\$, INPUT\$ - Input function	105
* HEX\$, OCT\$ - Hexadecimal and octal conversion	106
3. Special Functions	107
* POS, CSRLIN - Cursor functions	107
* POINT, SCREEN - Screen control function	108
* SPC, TAB - Functions in a PRINT statement	109
* TIME\$, TIME, DATE\$, DATE - Clock, calendar functions	110
* PEN - Light pen function	113
* ERR, ERL - Error-handling routine	114
* FRE - To check the usable memory area	114
* PEEK - To check the data in the memory	115

CHAPTER 7	:	GRAPHICS	Page 117
1.		BASIC Graphics	117
*		Normal mode and high resolution mode	117
*		WIDTH, SCREEN - Set graphic mode	117
*		Colour Code	120
*		PSET, PRESET - Placing the point and erasing it	120
*		COLOR - Colour designation	123
*		LINE - Drawing a line	124
*		Using the LINE statement in character mode	129
*		PAINT - Colouring figures	130
2.		Graphic Control by Machine Language	133
*		Level-3 VRAM structure	133
*		The structure of 5-bit colour information and colour register	134
*		Screen structure of high resolution mode	137
*		Colour character display in high resolution mode	141
CHAPTER 8	:	DISK BASIC MA-5300	143
*		Preparation	143
*		Start (Power on)	144
*		Difference between normal BASIC and disk BASIC	145
*		Usable memory capacity	147
CHAPTER 9	:	KEYGET	149
*		Input method without using return key	149
*		Further applications	153
*		Keyget for games	156
CHAPTER 10	:	GENERATING SOUNDS	160
*		Generating sounds with BASIC	160
*		Generating high tones with machine language	162
CHAPTER 11	:	USING THE LIGHT PEN	169
*		PEN function	169
*		Light pen interrupt	171
*		Light pen interface connector	173

CHAPTER 12 :	MASTERING CHARACTER STRING CALCULATION	Page 175
*	Addition of character strings	175
*	Character functions	176
*	Data compression and resolution	177
*	Compressing the data and storing it	177
*	Using the compressed data after resolving it	180
*	Making data base software	181
CHAPTER 13 :	USING THE PRINTER	188
*	Connecting the printer	188
*	OPEN, CLOSE - File open and close	189
*	PRINT#n, PRINT#n, USING - Output to the printer	190
*	Drawing on the printer's various functions	191
*	Printer page feed	191
*	Outputting the program list	194
CHAPTER 14 :	I/O SLOTS	195
*	Explanation of each terminal	196
CHAPTER 15 :	THE RS-232C INTERFACE	201
1.	RS-232C Interface Data Transfer	201
2.	RS-232C Interface Usage	202
*	Using Level-3 as a terminal (TERM command)	203
*	Connecting the Level-3 to a microcomputer	203
*	Connecting the Level-3 to a minicomputer	206
*	When Level-3 does not work as a terminal	207
3.	RS-232C Interface Program Operation	209
*	OPEN, CLOSE	209
*	INPUT#, PRINT#	210
*	RS-232C interrupt control	211
*	RS-232C connection to printer and other measuring equipment	213

SUPPLEMENTS

Page 218

Table S.1	Decimal .Binary .Hexadecimal .Octal Correspondence	214
Table S.2	I/O Map (&HFF00 - &HFFEF)	215
Table S.3	BASIC ROM Subroutine	215
Table S.4	Level-3 BASIC Work Area	216
	SAMPLE PROGRAMS	217
		238

BASIC MASTER LEVEL 3

MB-6890

CHAPTER ONE - OUTLINE

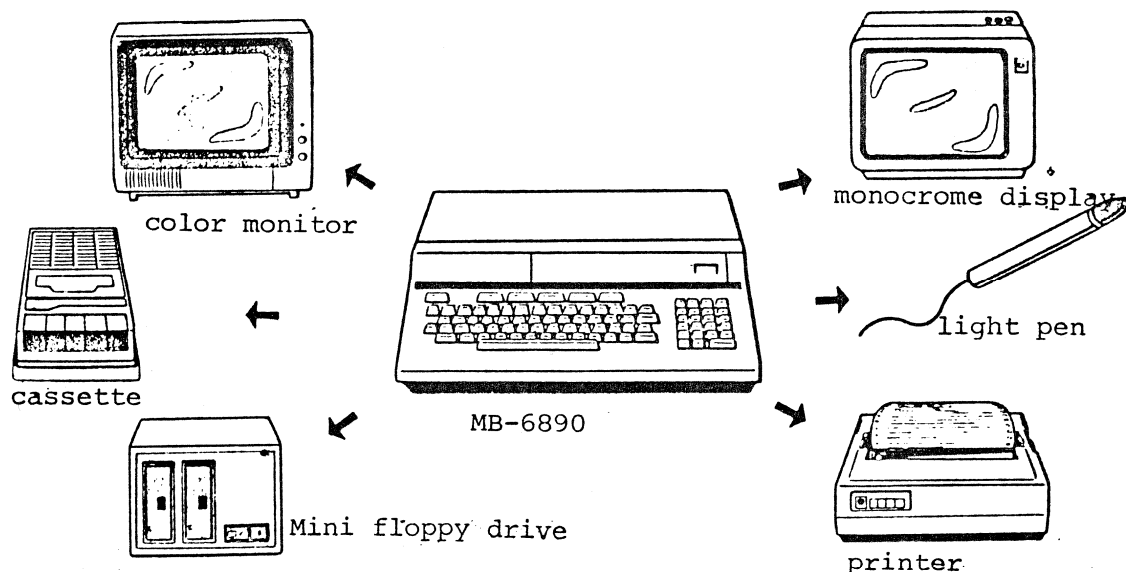
This introductory manual is written to help you understand correct MB6890 operation and the other manuals provided should be consulted to give you a clearer understanding.

1. LEVEL-3 CAPABILITIES AND KEY FUNCTIONS

The Hitachi Basic Master Level-3 is an outstanding personal computer with many functions.

This computer system, for which Extended BASIC is used to write programs, is capable of producing high resolution graphic display and is provided with built-in interface systems for connection with peripheral equipment to process a wide variety of jobs.

The multiple functions of the MB-6890 allow the system to respond to the versatile needs of business calculations, scientific and technical computations, programmer education, etc. The MB6890 is particularly suitable for small-scale business applications and for educational use in schools.



Features

1. High level programs can be readily produced by the use of Extended Basic, a highly functional language, which allows the programmer to use over 150 different commands.
2. This system is capable of providing colour graphics displays with a high degree of resolution - horizontal 640 dots x vertical 200 dots maximum.
3. This system can provide colour display in eight colours (six colours plus black and white). Each colour can be produced by specifying it in the program statement.
4. This system is equipped with ten separate numerical keys, which are convenient for the input of numerical data, and also with function keys which allow general purpose instructions to be input by a single key stroke.
5. It is possible to select either one of the two modes of display by means of a program statement, the 1,000-character mode (40 characters x 25 lines), which lends itself to easy reading, and the 2,000-character mode (80 characters x 25 lines).
6. This system is equipped with the HD 6809 (compatible with the Motorola MC 6809), which is now regarded as the ultimate in 8-bit CPUs.
7. This system has built-in interface systems for standard peripheral equipment, such as a printer.
8. An abundant array of peripheral equipment can be used in combination with this computer.

MB-6890 Hardware Specification

MPU - 6809 (8-bit parallel).

ROM	-	24KB standard (expandible up to 32KB) Masked ROM : Monitor and Basic
RAM	-	32KB standard (expandible up to 60KB) 5-bit x 16K (colour RAM)
Display	-	Horizontal 80 characters x vertical 25 lines / horizontal 40 characters x vertical 25 lines software selectable maximum 8 dots x 16 dots (interlace mode) 8 dots x 8 dots (non-interlace mode)
Display Characters	-	Characters and graphic symbols
Colour	-	8 colours (blue, red, magenta, green, cyan (light blue), yellow, white, black)
Graphics	-	640 x 200/320 x 200/160 x 100/80 x 100 dots Mixture with characters is possible 4 modes are selectable by software
Screen Control	-	Automatic scrolling, scroll window select, multipage (normal mode : Max. 16 pages High resolution mode : Max. 2 pages)
Keyboard	-	ASCII standard function keys, edit function keys and user-programmable soft keys are included. Keys provide audible feedback when operated.
Cassette Interface	-	600 baud FSK method
Light Pen Interface	-	Hitachi original
Video Interface	-	Colour : Separate colour video signal and separate synchronous signal Black & White : Composite signals

Printer - Centronics parallel
Interface

Communication- RS-232C interface
Interface 300, 600, 1200, 2400, 4800 baud

Expansion - 6 slots
Slot

Language - BASIC, machine language

Dimensions/- 45.0(W) x 12.5(H) x 51.5(D)cm
Weight - 7 kg

Power - AC100V 50/60 Hz

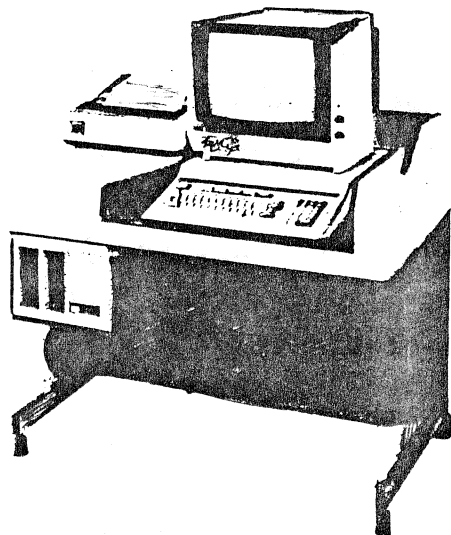


Photo 1.1 System Desk

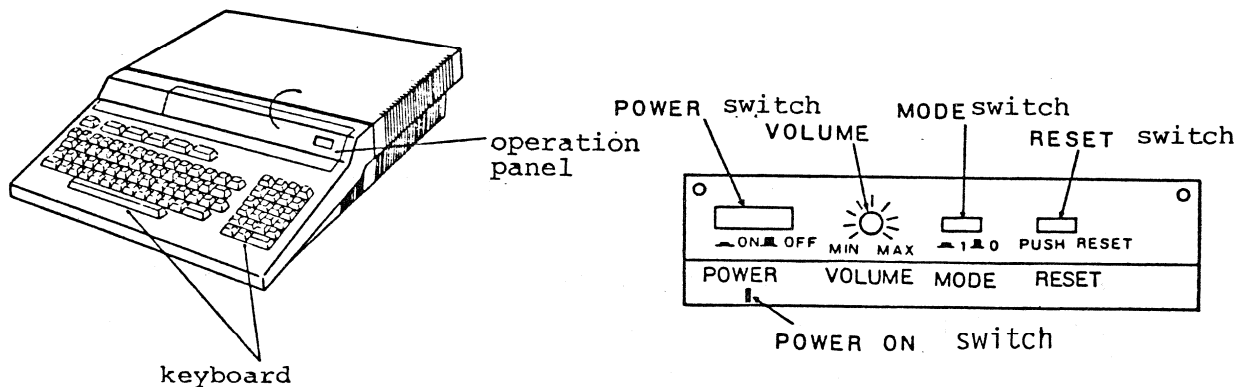
The front of the Level-3 consists of the keyboard and operation panel (Diag. 1.2 & 1.3).

Let us open the operation panel flap on the upper part of the keyboard. In there we have the POWER, VOLUME, MODE and RESET switches.

Usually the power switch and reset switch are situated on the back panel to avoid being operated by mistake. However, this

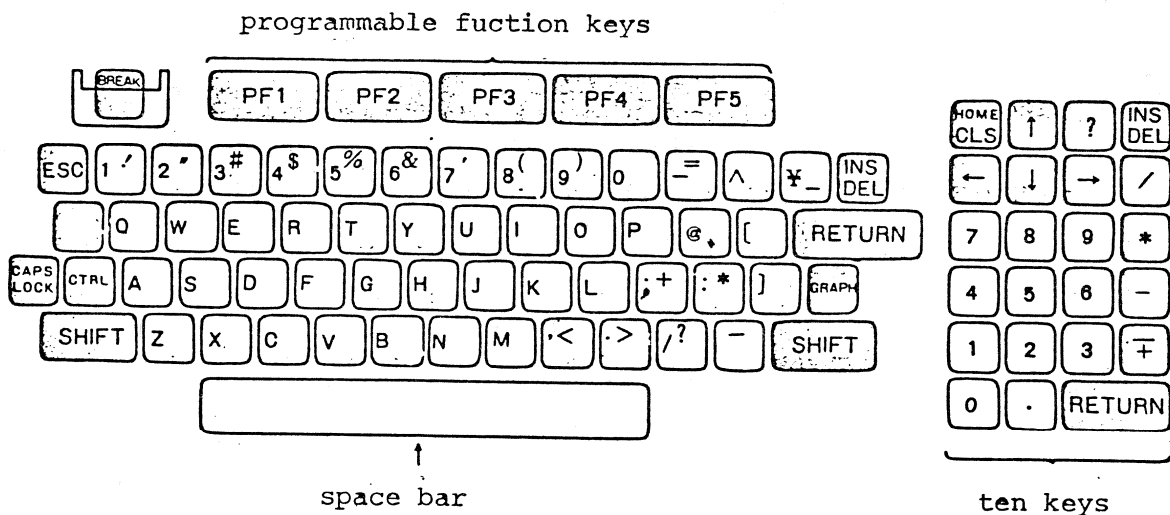
makes it rather difficult to operate the machine.

With the Level - 3 the operation panel is at the front so that turning the power on and off can be accomplished without such difficulty. Also being fixed under the flap there is virtually no worry of accidental operations.



Diag. 1.2 MB-6890

Diag. 1.3 Operation Panel



Diag. 1.4 Keyboard

Let us look at the keyboard next (Diag. 1.4). There are either white keys or grey/blue keys. The white ones are

general purpose characters and numerical keys and the grey (blue) ones are special keys which each have a special function.

The keyboard is divided into two parts. On the left side we find the ASCII keys and special keys, and, on the right side we have the [0] - [9] ten keys together with the cursor control keys. The keys duplicated on the left side and right side of the keyboard have the same function.

When inputting data with the numerical keys is frequent, the right-hand side keys are very convenient to use.

The Level-3 keyboard has incorporated human design philosophy to minimise fatigue from long hours of usage. The 'clicking' sound produced when depressing keys also helps prevent making typing errors.

Whereas most other personal computers use the soft scan method, the Level-3 has adopted the hardware scan method. This means that the Level-3 can store up to 30 characters in the input buffer which permits continuous fast type input.

An explanation of the special keys follows:

BREAK key: We normally press this key if we wish to discontinue the execution of the program. By pressing this key we are able to put the machine in the input waiting state.

This key also uses 'interrupt' so that it is possible to return to input state at any time. However, indiscreet use of this key may destroy the input data. Accordingly a plastic cover protects this key from mis-operation.

CTRL key: This key does not function if it is pressed alone, but used jointly with another key, it (control) can perform certain functions. For example, while depressing **CTRL** press the [G] key You

should hear a 'beep' sound. This is because the combination of **CTRL** + **G** has a bell function.

Another example : with the **CTRL** + **Y** key combination, the cursor moves one space to the right. This is the same as using the **→** key.

As shown with these two examples, many different functions are possible depending on the combination selected.

Table 1.5 Control Key Functions

No.	CTRL +	Function	Same Function Key
1	M	Return	RETURN
2	G	Bell sound	--
3	K	Cursor to home position	SHIFT + HOME CLS
4	^	Moves cursor one position up	↑
5	-	Moves cursor one position down	↓
6	Y	Moves cursor one position to right	→
7	J	Moves cursor one position to left	←
8	F	Moves cursor one word to right	SHIFT + →
9	B	Moves cursor one word to left	SHIFT + ←
10	R	Insert	SHIFT + INS DEL
11	H	Delete	INS DEL
12	E	Deletes one line after cursor	--

No.	<u>CTRL</u> +	Function	Same Function Key
13	<u>Z</u>	Deletes everything after cursor	--
14	<u>L</u>	Cancels and moves cursor to home position	<u>HOME</u> <u>CLS</u>
15	<u>T</u>	Sets horizontal tab	--
16	<u>Y</u>	Clears horizontal tab	--
17	<u>I</u>	Advances to next tab	--
18	<u>S</u>	Suspends screen output (Restarts with another key)	--
19	<u>C</u>	Breaks program execution (Restarts with CONT)	<u>BREAK</u>
20	<u>D</u>	Soft (hot) start	--

ESC key:
(escape)

With the Level-3, this key is not presently used and does not have any particular function. However, it is possible to give some function to this key if it is called for in the program.

CAPS
LOCK key:
(capital
lock key)

When this key is pressed the lamp directly below the key lights up red and the small alphabetical letters a - z can be input with the A - Z keys.

At this time, should you press the A - Z keys while depressing the SHIFT key, the capital letters A - Z will be displayed. When you press the CAPS
LOCK again, the lamp goes off and we return to the capital letter mode.

SHIFT key:

When this key is pressed jointly with another input key, the corresponding symbol of the

other mode is displayed. Under normal conditions (alphanumerical capital mode), if you push an input key together with the **SHIFT** key, small (lower case) letters are displayed.

There are two **SHIFT** keys, one of the right and one on the left. They both have the same function and it does not matter which key is used.

GRAPH key:
(Graphics)

Level-3 displays graphic characters in the non-interlace mode. Although they are not shown on the key top, each key has its corresponding graphic character as per Fig. 1.6. When you depress **GRAPH** + these keys together, the equivalent graphic symbol will be displayed.

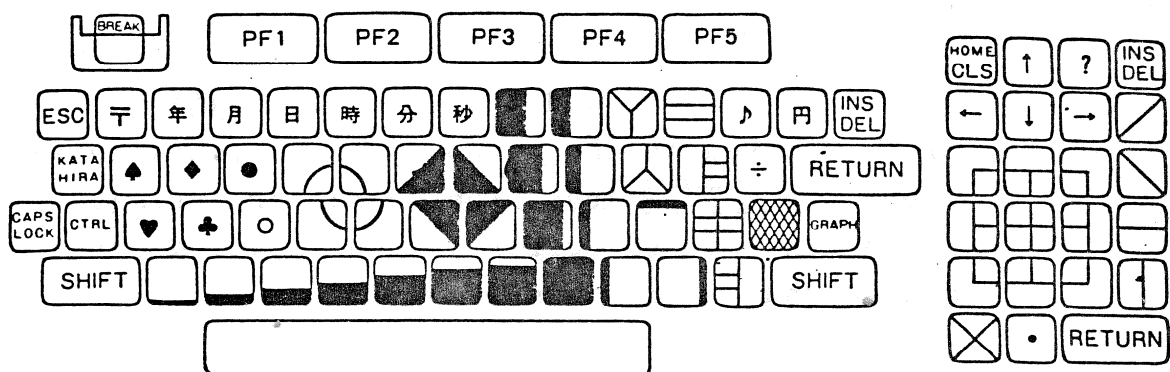


Fig. 1.6 Non-interlace mode graphic symbols

PF1 - **PF5**: By using one of these keys once we can obtain (Programmable the same result as having to operate many keys. function keys) .

Frequently used command words such as the following BASIC ones have already been set but you may register your own character strings freely.

Each key accepts a maximum of 15 characters which enables us to define any command in

BASIC. The variety of the character strings (in normal mode and shift mode) which we can define is a total of 10. (Note that PF6 - PF10 is in SHIFT mode.)

PF1 LOAD ^C R	PF6 TERM
PF2 ?DATE\$, TIME\$ ^C R	PF7 SCREEN
PF3 KEY	PF8 COLOUR
PF4 LIST ^C R	PF9 LIST"LPT0:" ^C R
PF5 RUN ^C R	PF10 CONT ^C R

RETURN key: This is the key to complete inputting one section of data. If you press **RETURN** the cursor moves to the head of the next line. The program and the data which is displayed on the screen is stored in a temporary memory - the key buffer. While the data is stored in this key buffer, correction of the key input and editing is possible.

Press the keys and the necessary data units are displayed on the screen. If they are correct, press the **RETURN** key. The character strings stored in the key buffer are then pulled out and stored in the main memory.

The **RETURN** key could be regarded as the key which actually handles the data when you communicate with the computer.

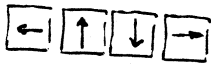
INS
DEL key:

(Insert/
delete)

This key can perform two actions if it is used with the **SHIFT** key. If you press **INS**
DEL the letter or numeral directly in front of the cursor is deleted.

With **SHIFT** + **INS**
DEL combined, a space is inserted just before the cursor. There is an **INS**
DEL key on both the left and right-hand side of the keyboard. As they are exactly the same, use

whichever is convenient.



Cursor Control keys.

With these keys you can easily move the cursor in the direction of the arrow. If you press one of these keys while holding down the **SHIFT** key, the cursor moves continuously one space at a time. It is useful to move the cursor long distances.



key:

(Home clear)

When this key is pressed, the cursor returns to the home position at the top left-hand corner of the screen and the screen is cleared at the same time.

If you press **SHIFT** + **HOME CLS**, the cursor returns to the home position without clearing the screen.



key:

(Print key/
arithmetical
operations
key)

? is an abbreviation of the PRINT statement. By using **?** with **0**-**9**, **+**, **-**, *****, **/**, **(**, **)** simple arithmetical calculations can be performed.

Example: ?186*4+200
 944

Repeat
Function:

Level-3 keys have a repeat function after one second, so the same character is displayed repeatedly. This function is particularly useful when we need to move the cursor. There is no repeat function with the function keys.

2. SCREEN MODE CHANGE

* Interlace mode and Non-Interlace Mode

The Level-3 has two screen display modes. In interlace mode, to form one screen, or picture, there is a double screening. This means an 8 x 16 dot character can be

displayed the same size as an 8 x 8 dot character. With this new feature, the L-3 character generator is twice as big as a normal system.

The graphic characters are displayed only in interlace mode. To switch from one mode to the other we use a SCREEN command.

```
SCREEN,, 0      interlace mode
SCREEN,, 1      non-interlace mode
```

* Graphic Mode

Level-3 has four graphic modes.

Graphic Mode	{	Quality Res. Mode	{	Normal Mode
			{	High Res. Mode
	{	No. of Horizontal Display Characters	{	40 Characters
				80 Characters

Normal mode displays characters only. In this mode, only the characters set in the character generator can be displayed on the screen. However, compared with the high resolution mode, this mode requires fewer bytes to form one screen. (This means that there are more memory bytes able to be used for the program than with high resolution mode.)

When you set 40 characters as the number of horizontal display characters, 40×25 lines = 1000 characters can be displayed on the screen. On the normal mode screen 1 byte is allocated to one character. Therefore, 1000 bytes (1Kbyte) memory is in use. In the case of 80 character mode, this is $80 \times 25 = 2000$ character/screen, so 2Kbyte memory is in use.

With high resolution mode each dot, which forms the character, can be specified. So, quite complicated diagrams can be drawn.

The resolution of horizontal display characters is 320 dots in the 40 character mode and 640 dots in the 80 character mode. We reach this figure by way of the fact that each character is 8 dots wide.

$$8 \times 40 = 320 \quad (40 \text{ ch. mode})$$

$$8 \times 80 = 640 \quad (80 \text{ ch. mode})$$

Vertically speaking, characters in both the interlace and non-interlace mode have 8 dots. Both 40 ch. and 80 ch. mode are displayed in 25 lines so the vertical resolution becomes $8 \times 25 = 200$ dots.

Thus with high resolution mode the following graphics can be obtained:

40 ch. mode 320 x 200 dots

80 ch. mode 640 x 200 dots

Let us check the memory capacity necessary to make one screen in high resolution mode.

In this mode, each dot is equivalent to a bit. In the horizontal direction, each byte can display 8 dots, therefore 320 dots means 40 bytes and 640 dots means 80 bytes of memory is occupied. Against this, the vertical resolution is 200 bits so that to show the lines in both 40 byte and 80 byte memory we need 200 lines. (See Fig.1.7)

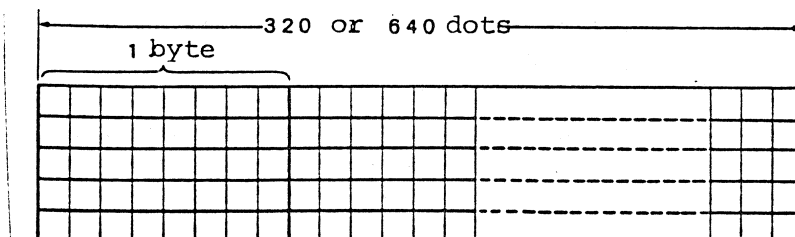


Fig. 1.7 High Res. Mode
Horizontal Dot
Resolution

0	4
1	5
2	6
3	7

Fig. 1.8 Normal Mode
Character
Resolution

As a result, where the horizontal display is 320 dots a memory capacity of $40 \times 200 = 8000$ (or 8Kbytes) is required and for 640 dots, we need $80 \times 200 = 16,000$ (or 16Kbytes). (You can understand that we require much more memory than with the normal mode which we calculated before.)

So, bit image graphics are obtained by using high resolution mode but even using normal mode we can make simple graphics.

If the character is separated into eight segments as shown in Fig. 1.8, a simple graphic of 80×100 dots in 40 ch. mode and 160×100 dots in 80 ch. mode can be shown. In this case, the memory is significantly saved compared with high resolution mode.

* Colour Display

Level-3 has byte control colour mode. Horizontally a colour resolution of 40 for the 40 ch. mode and 80 for the 80 ch. mode is possible. Vertically, you can obtain a colour resolution of 200 in high resolution mode. However for normal mode it is 25. This is because each colour segment is divided into byte (character) units.

Table 1.9 Graphic Mode Summary

Fineness Mode	Normal Mode		High Resolution Mode	
	40 columns	80 columns	40 columns	80 columns
No. of horizontal display characters				
Graphic Mode (Hor. x Vert.)	80×100 dots	160×100	320×200	640×200
Colour Res. (Hor. x Vert.)	40×25 dots	80×25	40×200	80×200
Memory Capacity On One Screen	1K byte	2K	8K	16K

Fineness Mode	Normal Mode		High Resolution Mode	
Character modes concurrently useable	40ch. x 25 l.	80 x 25	40 x 25	80 x 25
User Area (RAM 32 KB)	30 K bytes	29 K bytes	23 K bytes	15 K bytes

* Multi-Page

The Level-3 has a multi-page function. This means we can call up a particular page from among the many stored and, by changing the page very fast, we can achieve a simple animation effect.

The number of the pages is decided by the memory capacity. The Level-3 has a maximum of 16 byte VRAM area. The VRAM memory capacity is set by the MODE switch when the power is turned on or by using the BASIC command "NEW ON". If you divide the VRAM memory capacity by the screen capacity of your desired display, you can calculate the number of pages allowed by that memory capacity.

Example: We set the screen for high resolution mode. The VRAM memory capacity is 16 Kbytes. Let us try the multipage function. We should use 80 ch. normal mode. The memory capacity of one page is 2Kbytes therefore $16/2 = 8$, and we get 8 pages.

Table 1.10 Multipage

→ Screen for Multipage					
Orig Set Screen	M/Page Screen	40 ch. Normal Mode	80 ch. Normal Mode	40 ch. High Res. Mode	80 ch. High Res. Mode
	40 ch. Normal	1	X	X	X
	80 ch. Normal	2	1	X	X
	40 ch. High Res.	8	4	1	X

80 ch. High Res.	16	8	2	1
---------------------	----	---	---	---

(X = cannot be used)

If we change 80 ch. high resolution mode to 40 ch. normal mode, we get $16/1 = 16$ (16 pages) and, if we use 40 ch. high resolution mode we get $16/8 = 2$ (2 pages). (See Table 1.10)

* Dip Switch Setting

Refer to the MB-6890 Level-3 Basic Manual.

3. CONNECTION TO PERIPHERALS

The Level-3 system can be expanded by connecting various options.

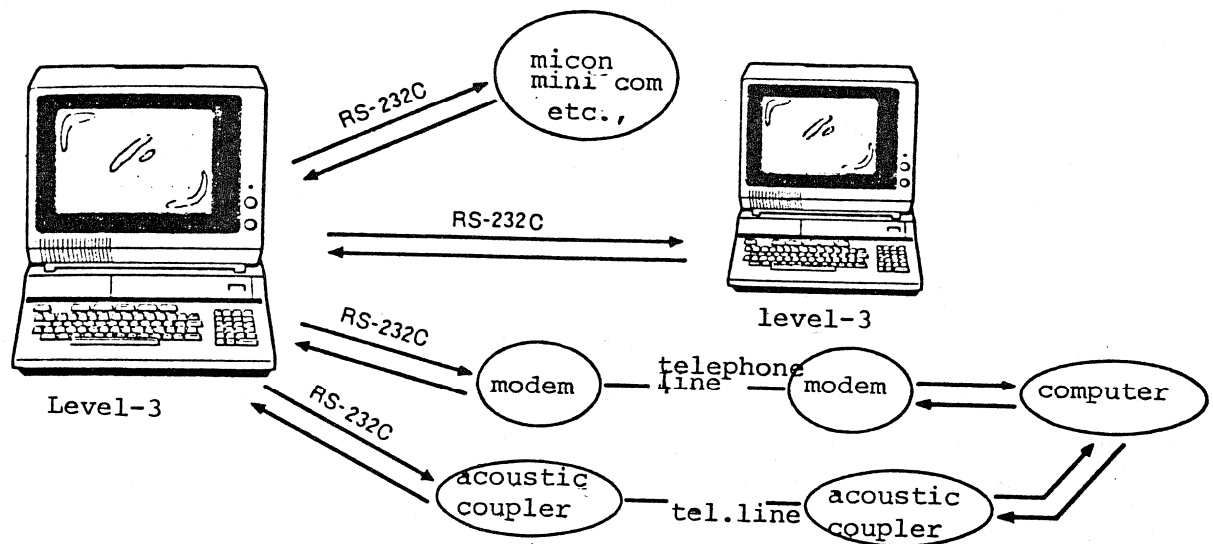
The standard unit has cassette tape recorder, printer, light pen and RS232C interfaces. The most interesting accessory is the light pen. The control command of the light pen is in BASIC ROM. The resolution of the light pen is one byte. This is the same whether the Level-3 is used in high resolution mode or normal mode, and, it has an 80 x 25 matrix in the 80 ch. mode and a 40 x 25 matrix in the 40 ch. mode.

The RS232C connector is for communicating with other computers which have an RS232C port. A simple example would be to connect one Level-3 to another Level-3 through the RS232C port, and set both to the terminal mode. Then if you press the key on one Level-3, the character, that you press the key for, appears on the screen of the other Level-3. From this we know that correct data exchange has been effected between the two computers.

To connect up with other equipment we can either use a direct cable or we can use modem, or even an acoustic coupler.

Modem is used to plug into Telecom lines. An acoustic coupler may also be used with telephone lines. Whereas Modem sends the signal as is, normally the coupler converts the signal into sound first.

In all, this RS232C interface is an extremely useful port if used with equipment which is provided with the same port.



Diag. 1.11 Communication with Level-3 and other equipment with the RS232C.

4. EXPANDABILITY OF LEVEL-3

On top of the Level-3 having built-in interfaces it also includes six card connectors or slots.

The signals from the Level-3 are supplied to each slot so by inserting a card in these slots the required signals are automatically output.

All the slots are inside the cabinet on the main PCB, which makes for compact expansion and high reliability. Most of the expansion cards presently available are made to fit these slots and even when installed, they look as if they were built in from the start.

When the signals are taken out from the interface card, the connector is fixed to the card edge. The interface card is not only fixed by the slot, but also by a plastic guard thus preventing disconnection of the cards due to vibration. Furthermore, as the slot is situated on the corner of the main board, big interface cards may also be used.

Expansion cards for floppy disk drives as well as for music sound generator cards, standard I/O are available and new cards will also be released in the future.

Apart from this expansion interface slot, the Level-3 also has two RAM expansion connectors. The RAM memory expansion card permits two boards each with 16K bytes to be used. The memory map is shown in Fig. 1.12. Refer to 16K RAM card specification for more details. As well as six slot expansion interfaces the Level-3 has two RAM expansion slots so that even if the memory is expanded to its full capacity there are another independent six interfaces that may be used.

0000	work area	work area	work area
	screen area	screen area	screen area
4400	user area	user area	user area
7FFF	free area	8 k byte	
A000	ROM area BASIC Monitor	ROM area BASIC Monitor	extended 32k bytes
FF00	I/O	I/O	I/O
FFFF			
	standard RAM-32k	RAM card 1, RAM-40k	RAM card 2pcs, RAM-60k

Fig. 1.12 . Memory Map with Expansion RAM Card

CHAPTER TWO

LEVEL-3 BASIC

This chapter outlines easy basic language. If you are going to operate the Level-3, you will need to understand BASIC statements.

To input the program, press the keys in this order:

NUMBER BASIC STATEMENT RETURN

The line number should always be placed at the head of each line. It shows the order of the statements and is extremely important. The numbers should be in consecutive ascending order and are normally used in intervals of 10 - 10, 20, 30 Of course, 1, 2, 3 ... numbering could be used. But, if later you needed to insert a new line number in the program you would have to retype the statements again merely to add one in the middle. So it is recommended to leave some line number room between lines.

Level-3 BASIC has many statements but nine statements are sufficient for a simple program.

Let us memorise these statements. The best way to do this is to actually press the keys and execute each statement. If the input program is somehow incorrect an ERROR message will appear on the screen together with the line number in which the mistake occurs. The error message will always come up no matter how many times the mistake is repeated so you can operate the keys with peace of mind. The error message is only a friendly message to you.

Now, let us move on to the nine fundamental statements.

* PRINT (Output to the Screen)

As you know, every microcomputer has a keyboard and a monitor. These pieces of equipment become the means of

communication between the microcomputer and us.

If you press any input key, the character which corresponds to the key appears on the screen. Now, let us carry out the same operation using a statement. If you want to output something on the screen, the PRINT statement is used. Let us type the example below without worrying about the line number:

```
PRINT "LEVEL 3"
```

The (")(double quotation marks) is made by pressing SHIFT + 2 on the full keyboard (not the 2 on the special ten keys located on the right side). Do not forget to press RETURN after this. We operate RETURN to tell the computer "this is the end" then the instructions will be executed.

```
PRINT"LEVEL3"  
LEVEL3
```

When you pressed RETURN, immediately the characters LEVEL 3 appeared on the screen, didn't they? The PRINT statement ensures that the characters are printed on the screen as specified.

Now look at the screen once more. We input LEVEL 3 between two sets of " but the displayed characters do not have the ". This means PRINT displays all the characters sandwiched by the quotes.

This is the first function of PRINT. Now we continue with these:

```
PRINT"HITACHI"  
HITACHI
```

```
PRINT"BASIC MASTER"  
BASIC MASTER
```

```
PRINT"LEVEL3"
LEVEL3
```

Writing characters is easy as we have seen. Now look at the next sample programs :

```
PRINT"123"
123
```

```
PRINT 123
 123
```

Both are displayed the same except that the second one has a space just in front of the 123. This means the + sign has been omitted. In fact, the first statement displays the character string called '123' but the next is showing the numeral 123. Except for 0, all real numbers have a + or - sign, therefore if you display numbers, we get a space produced by leaving out the + symbol, or a - sign.

Example:

```
PRINT 100+345
 445
```

This output 445 is the result of $100 + 345$. By this, it is obvious that numerical calculation can also be performed. So, PRINT has the function of outputting numbers and displaying the result of calculations. This is the second function of PRINT.

Let us do a more complicated calculation. For multiplication we use an asterisk (*) instead of an (x) and a (/)(slash) for division instead of the (\div) symbol. If we calculate $12 + 32 \div 4 - 3 \sim$

```
PRINT 12+32/4-3
 17
```

In Basic language, the priority mathematical calculations are

multiplying and dividing over adding and subtracting. Let's calculate the expression with a ():

```
PRINT (12+32)/4-3
```

8

The calculation inside () is carried out first. This is also according to operation sequence.

In Basic, if the numerical expression is written after the PRINT, the calculation will be performed exactly as per the formula - rather like a calculator.

To input the statement without a line number is called direct mode execution and it is useful for simple calculations. But some statements are not suited to direct mode execution. You may like to try which statements are and which are not suited. If incorrect, the ERROR message will appear.

This direct mode execution does not make anywhere near full use of the computer's ability. There is another mode called program mode execution. Let's try it.

We write the line number at the beginning so that the program can memorise it.

```
10 PRINT"HITACHI"
```

This time, it is not executed immediately. However this line is stored in the memory. Let's continue the programming. As mentioned before, we use the line numbers in increments of 10.

```
10 PRINT"HITACHI"
```

```
20 PRINT"BASIC MASTER"
```

```
30 PRINT"LEVEL3"
```

```
40 PRINT"MB-6890"
```

Now all the program from line number 20 onwards is also stored in the memory. Let's confirm that.

To display the program in the memory we use the LIST statement. LIST is the statement to output the program list onto the screen. Press LIST **RETURN** (or **PF4**)

LIST

```
10 PRINT"HITACHI"  
20 PRINT"BASIC MASTER"  
30 PRINT"LEVEL3"  
40 PRINT"MB- 6890"
```

In this way, the stored program is displayed in line number order and we can confirm that it is correct. The LIST statement is used to correct, to change and to check the program. You can use this statement any time you want to see the program. We have used the word "statement" but correctly it should be called a command because it relates directly to the computer.

Let's execute the above program next.

In direct mode execution the program was executed immediately after the key input but this time, we must give the computer the command to execute the program. This command is called RUN.

Press RUN **RETURN** (or **PF5**) and the program is executed forthwith:

```
RUN  
HITACHI  
BASIC MASTER  
LEVEL3  
MB- 6890
```

If there is a grammatical error such as a spelling mistake, the 'Syntax Error' message appears on the screen. If this happens, what procedure is required?

First of all, display the program in memory with the LIST command. Then correct the line which is indicated in the error message. The quickest way is to replace the incorrect line with the new correct line. This method means inputting the same line number with the newly corrected program. If the same line number is input, the previous line number program is ignored and the new input program becomes effective. Even if the new program becomes longer than the previous one, it does not matter.

Next, let's try moving a character to the right using the PRINT command. To move a displayed character, we have the function of TAB(n). This TAB moves the cursor a specified number of locations decided by n to the right. In direct mode:

```
PRINT TAB(10);"COMPUTER"
      COMPUTER
```

The character COMPUTER is displayed ten character spaces from the left hand edge. The semicolon (;) in the above PRINT sentence has the function of writing numerals and characters without spaces. In this example, the character COMPUTER is displayed after ten character spaces.

Let's confirm the function of (;) with another example:

```
PRINT "X=";45*10
X= 450
```

```
PRINT "HUDSON";TAB(15);"SOFT"
HUDSON          SOFT
```

This is the same in program mode.

The PRINT statement is used so frequently that we have an abbreviation of one character - ? - to replace it.

```
10 ?"LEVEL3"  
20 ?"HITACHI"
```

```
RUN  
LEVEL3  
HITACHI
```

If LIST is pressed, ? changes to PRINT when it is displayed on the screen.

```
LIST
```

```
10 PRINT"LEVEL3"  
20 PRINT"HITACHI"
```

* VARIABLES * SUBSTITUTES

Variables can be regarded as the places where values used in BASIC programs are stored. Each variable is given a name.

Example:

```
10 A=5  
20 B=3*A  
30 PRINT A,B
```

This program means put 5 in place of the variable A. Put the product of the multiplication of 3 and A in the variable B. Write the value of A and B.

If this program is executed the value of A and B is displayed consecutively on the screen.

```
RUN  
5          15
```

The above A and B are called variable names. The variable name can be used if it complies with the following rules.

1. It must be an alphanumerical with an alphabetical head.
2. No symbols such as +, -, * or / should be included.
3. Only names of up to 16 characters may be used.
4. No BASIC commands can be used in the head.

A, ATOB, MLIST ~ are acceptable as variables

1A, TOAB, LIST3, A+B ~ cannot be defined as variables

There are other variables but these will be discussed later.

Stipulations when making Substitutions -

Putting a number for a variable is called substituting. This must obey the following rules:

1. Do not write constants or numerical formulae on the left side, only variables.
2. On the right side, constants, numerical formulae and variables can be written but, variables on the right side must be defined before being used. (If not defined, they are regarded as being 0.)

Let's think about the example we used before -

A=5

B=3*A

This substitute sentence means to substitute 5 for A and B for the product of the multiplication of 3 and A. In the second line, as A has already been defined as 5, the value of 15 (3 x 5) is substituted for B.

Next example -

K=0

K=K+1

Mathematically, it is a rather incomprehensible formula, isn't it? However, the meaning of the symbol (=) in a program differs

to its mathematical one. In this case, (=) is used to substitute the value on the right side with the variable on the left. It is probably easier to think of (=) having the same meaning as the symbol \leftarrow

A \leftarrow 5
B \leftarrow 3*A

K \leftarrow 0
K \leftarrow K + 1

Let's now input 111 and 222 for the variables A and B and substitute the sum of A and B for variable C

Before doing that though, we must erase the old program on the screen. One way to do so is to turn off the power but a much better way is to use the NEW command. Using this command we press NEW and then RETURN and the memory program is erased. To make sure, press LIST-

NEW

Ready

LIST

Ready

You can see the program has been erased. Let's now execute the new program.

```
10 A=111
20 B=222
30 C=A+B
40 PRINT"A+B=";C
```

RUN

A+B= 333

* INPUT - Inputting data to the variables from the keyboard

INPUT is the command to enter data from the keyboard to the designated variable. We cannot use direct execution with

this statement. The format is as follows:

INPUT n (n is a variable name)

Press NEW and enter the following program -

```
10 INPUT A
20 INPUT B
30 C=A+B
40 D=A- B
50 E=A*B
60 F=A/B
70 PRINT"+";C;" - ";D
80 PRINT"*";E;" / ";F
```

Once entered correctly, execute -

RUN

?

After RUN, the INPUT statement is executed, a question mark (?) is output and the cursor is blinking as the BASIC waits for input from the keyboard. Press 10 and RETURN. The second INPUT statement is then executed and the ? is again output. We input 5 -

RUN

?10

?5

+ 15 - 5

* 50 / 2

Let's look a little more closely at the PRINT statement on line numbers 70 and 80. C, D, E, F are written without ("). This shows that the PRINT statement outputs the content of the variable if the variable name is written directly.

* GOTO - Changing the Program Flow

The programs so far have all only flown from top to bottom. However, this does not allow for the same calculations to be repeated using the same procedure. It is at these times that the GOTO statement is so useful and we explain it here.

We write the GOTO statement as -

GOTO n - n is the designated line number
and can go to any line

Let's add

```
90 GOTO 10
```

to our program above.

Press RUN, input the data for A and B. The result is displayed and Level-3 awaits further input. And keeps in the input wait condition, doesn't it? The GOTO statement makes the program flow in a circle and this is called a LOOP.

The rules for using the GOTO statement are as follows:

1. Variables and formulae should not be written after GOTO.
(They must be integers.)
2. The line number defined by GOTO must exist in the program.

With GOTO the program runs forever, but we must stop the program somewhere. To do this, press BREAK. If you want to re-start the program, input RUN.

* IF - Changing the program flow

We can change the program flow with a GOTO statement but this is an unconditional statement and there are times when you want to set certain conditions to control the program flow. This is when we use the IF statement.

The format is -

IF arithmetical expression THEN any statement(or line number)

If the condition is met the THEN portion of the statement is executed. If the condition is not met the computer will skip this instruction and execute the next line.

Example -

IF A < 10 THEN ---

This means "if variable A is smaller than 10 ..."

After THEN, almost any statement could be written.

IF -- THEN PRINT "START"

IF -- THEN INPUT B

IF -- THEN GOTO 40

The GOTO statement is used so often after THEN that it can be omitted.

IF -- THEN 40

Or, you can omit THEN if a GOTO sentence follows -

IF -- GOTO 40

We normally use a flow chart if the program becomes complicated.

Question : Decide whether input numbers are even or odd.

Display and repeat this five times.

We then construct the program according to the flow chart.

See Diag. 2.1.

10 K=0

20 INPUT A

```

30 B=A-INT(A/2)*2
40 IF B=0 THEN 90
50 PRINT"ODD"
60 K=K+1
70 IF K<>5 THEN 20
80 END
90 PRINT"EVEN"
100 GOTO 60

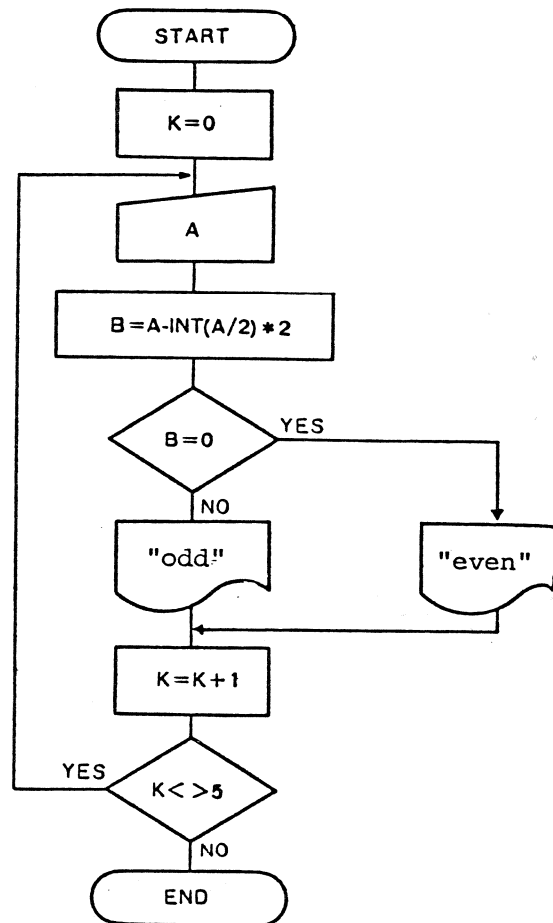
```

We should consider how this program was made.

There are two points with this problem. The first is to decide whether the input numbers are odd or even and we need to make a formula for this.

```
30 B=A-INT(A/2)*2
```

The $\text{INT}(A/2)$ is to make integers. For example, $\text{INT}(1.5)$ becomes 1 and $\text{INT}(5.2353)$ becomes 5. With negative numbers $\text{INT}(-3.345)$ becomes -4 (not -3) and $\text{INT}(-0.135)$ becomes -1.



Diag. 2.1 Flow Chart

Back to the program. If the input number is even, $A/2$ becomes an integer with the same value as $\text{INT}(A/2)$. Therefore, $\text{INT}(A/2) \times 2$ becomes the same as A and B is 0.

If the input number is odd, $A/2$ becomes a decimal, so that $\text{INT}(A/2) \times 2$ does not have the same value as $\text{INT}(A/2)$ (it has become a decimal). As a result, the value of B is not 0. Line number 30 contains the information on whether the input number is even or odd.

And this will be judged by the IF statement in line number 40.

```
40 IF B=0 THEN 90
```

The second point in the problem calls for the program to be repeated five times. This means making five loops.

Look at $K=K+1$ in Line 60. As this means $K \leftarrow K+1$, at every execution of this line number, the value of the K increases by 1. To accomplish this we set K as 0 in Line 10. We then judge if the K is 5 or not in Line 70 with an IF statement. If $K \neq 5$, the program goes to line number 20 and if $K=5$, the program goes to line number 80 and stops the execution.

The END statement stops the program. If no other program follows it may be omitted. To start execution again, input RUN.

* FOR - NEXT LOOP - Repeat Jobs

A loop is used to repeat the same program. We did five loops in the previous program using the IF statement. However, this means we have to follow several steps such as adding one at a time, judging the conditions, and deciding the next line number which can all be very troublesome.

In BASIC we have a statement especially for these loops, and that is the FOR and NEXT combination. Each FOR must have a matching NEXT.

Let's rewrite the previous program using this new statement.

```
10 FOR K=1 TO 5
20 INPUT A
30 B=A-INT(A/2)*2
40 IF B=0 THEN 60
50 PRINT"ODD"
55 GOTO 70
```

```

60 PRINT"EVEN"
70 NEXT K
80 END

```

There are programs which require IF statements but if the loop number is known from the start, a FOR-NEXT statement is normally used.

Generally the FOR statement is shown as:

```
FOR v=n TO  $n_1$  STEP  $n_2$ 
```

v is the loop variable; n is the starting value; n_1 is the end value and n_2 is the increase amount. The STEP statement may be omitted if the increase amount is only 1.

Example -

```
FOR I = 1 TO 20 STEP 1
```

This FOR statement designates increasing from 1 to 20 at intervals of 1. We can rewrite the above by omitting the STEP statement like this -

```
FOR I = 1 TO 20
```

Of course, if the increase amount is 4, we must write -

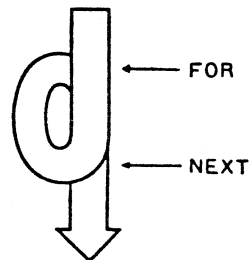
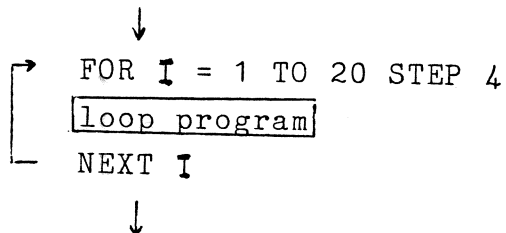
```
FOR I = 1 TO 20 STEP 4
```

In loop programs, the FOR statement looks after the first line. To watch over the last line we have the NEXT statement. We write this -

```
NEXT v (v is the loop variable)
```

After the first loop, the NEXT statement is found, STEP n_2 is added to the variable v, which changes to the new number, and we return to FOR and a second loop is executed. This

continues until the variable *v* is over the limit. When this happens, the program goes on through (out of the loop) and executes the next line.



We call this kind of loop a FOR-NEXT loop.

Example -

Diag. 2.2 FOR-NEXT loop

```

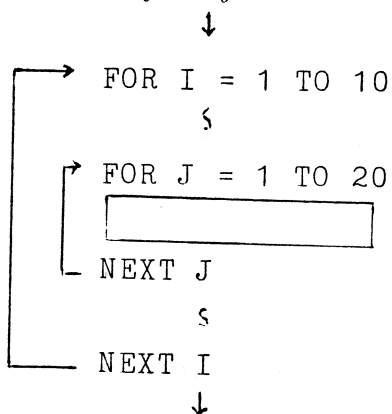
10 FOR I=1 TO 10
20 PRINT I;
30 NEXT I
  
```

RUN

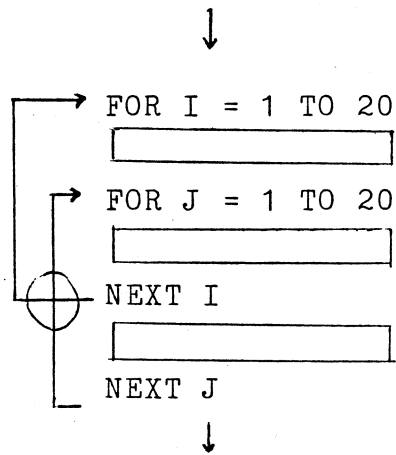
```

1  2  3  4  5  6  7  8  9  10
  
```

There are really no restrictions to using FOR-NEXT loops in your program as long as you do not destroy the loop. That is, they may nest but may not cross.



The following program will not work as the loops are destroyed.



So, just organising these last points -

- * We can repeat for a designated number of times a program which is contained in a FOR and NEXT loop.
- * The program loop must be completed in that loop.

Example -

```

10  FOR I=0 TO 10
20  FOR J=0 TO 10
30  PRINT I*J;
40  NEXT J
50  PRINT
60  NEXT I

```

* GOSUB-RETURN - Subroutine

IF statements and FOR-NEXT loops can execute the same program many times. But, this statement cannot be used if the loop is required at many parts of the program.

For this kind of situation, we make a subprogram which is a part of the program to be used many times. We can call the subprogram into the main program whenever it is required. We call this sub program a subroutine and in BASIC we can set this with a GOSUB statement.

As shown in Fig. 2.3, the subroutine is written in after the main program and it ends with a RETURN statement.

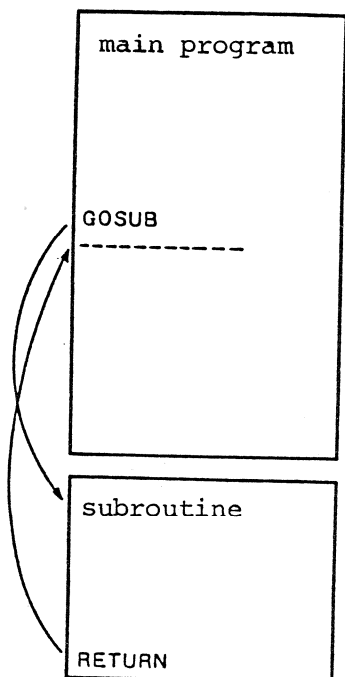


Fig. 2.3 Main Program
& Subroutine

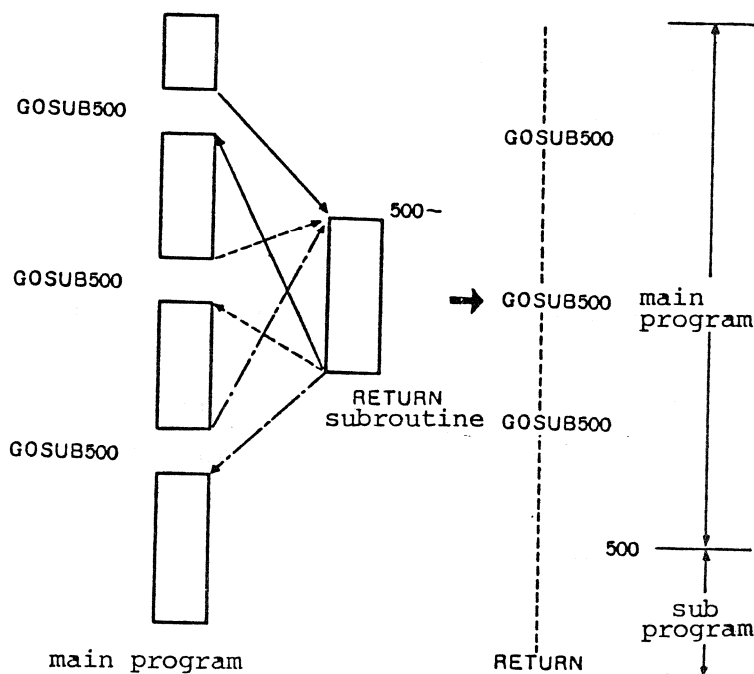


Fig. 2.4 Subroutine & Program
Flow

When the subroutine is called up from the main program, the first line of the subroutine *n* is written after GOSUB.

GOSUB *n*

For example, if the subroutine is to start from line number 1000, this is displayed -

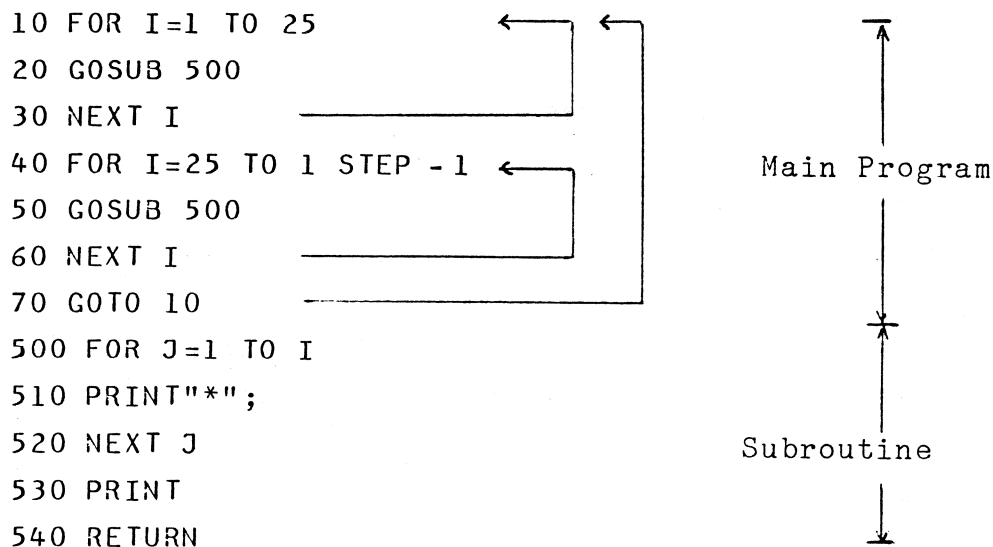
GOSUB 1000

Let's now look at how the program flows when the GOSUB is executed.

GOSUB is different to GOTO in that it always comes back to the main program. When the program is branched to the subroutine by GOSUB, the line number of the GOSUB is stored. After the execution of the subroutine, the computer comes back to the line that follows the GOSUB statement in the main program

because of the RETURN statement. If there is another GOSUB, the program executes the new subroutine and consequently comes back to the main program again.

Here is an example of a program using GOSUB -



The portion of the program from line number 10 to 70 is the main program and from 500 to 540 is the subroutine. If the line numbers of the main program and the sub program are separated widely as in the example, they can be understood clearly.

There are two FOR-NEXT loops in the main program. Line numbers 10 to 30 designate an increase in intervals of 1 from 1 to 25. Line numbers 40 to 60 form the loop to reduce 25 to 1 one at a time. There is a subroutine call command in each of these two loops. There is also a FOR-NEXT loop in the subroutine.

The FOR statement in line number 500 obeys the loop variables in the main program. The (;) in line number 510 is to write (*) next without a space between, and, PRINT at 530 is to stop (*) being packed by (;).

* SUMMARY OF THE BASIC STATEMENTS & COMMANDS

We summarise here the statements which have been described so

far.

* INPUT/OUTPUT STATEMENTS

INPUT ... Input from the keyboard

INPUT A

PRINT ... Output to the screen

PRINT "ABC" ; " - "

PRINT 123

(; is used to write numerical values and letters without intervening spaces)

* JUMP STATEMENTS

GOTO ... Jump

GOTO 300

GOSUB ... Subroutine call

GOSUB 500

RETURN ... Return from subroutine

(GOSUB and RETURN are pair statements)

* LOOP STATEMENTS

FOR ... Set loop limits

FOR	I	=	1	TO	10	STEP	2
	↑		↑		↑		↑
	loop		starting		end		interval
	variable		value		value		

NEXT ... Shows end of loop

(These are also pair statements)

* PROGRAM FLOW CONTROL STATEMENTS

```
IF ...    IF A<> 0 THEN 500
          IF A> 0 GOTO 300
          IF A>= B THEN PRINT "HIT"
```

```
SUBSTITUTE ...  A = 10
                J = K + 1
                X = Y
                ↑   ↑
            variable numerical formula, formula containing
                                variable
```

These are the basic nine statements in a program. As well as these we also have commands direct to the microcomputer -

```
LIST ... Output the program stored in the memory to the
        screen
```

```
RUN ... Execute the program
```

```
NEW ... Erase the program
```

Important Points to Remember when you Write the Program:

The line number is required if you want the computer to execute the program in program mode. If no line number, the program is executed in direct mode.

```
10 PRINT "LEVEL3"
```

When a statement is later inserted between two existing line numbers the line number of this newly inserted statement should lie between these two line numbers.

```
10 PRINT A
20 A=C+D
```

If you insert $C = 30$ between the above two statements, input
15 $C=30$ then **RETURN** so that we have -

```
10 PRINT A
15 C=30
20 A=C+D
```

To erase one line, input only that line number then **RETURN**.

```
10 PRINT A
15 C=30
20 A=C+D
```

If line number 20 is to be erased, input 20 then **RETURN**.

```
10 PRINT A
15 C=30
```

CHAPTER THREE

POWER ON

Let's turn the power on.

First, open the operation panel flap and push the power switch on. Now BASIC is called up and the screen display shows the title and memory capacity that we can use. See Photo 3.1.

As mentioned previously, the Level-3 memory area that can be used varies according to the screen mode. And, depending on that, how much memory area you can use is decided. The total area, that is the screen area and the user area, is about 31 Kbytes. So the area you can use is 31 Kbytes less the specified screen area.

Table 3.2 shows the screen mode occupancy (VRAM).

The dip switch has been set to high resolution mode when delivered. The mode switch is the 40-character/80-character change switch. The display shows that 22378 bytes (40-ch.) or 14186 bytes (80-ch.) of memory is open to the user. Photo 3.1 shows the 40-ch. high resolution mode.

If the power is switched on at 40-ch. high resolution mode, and you try to set it in 80-ch. high resolution mode, this causes an error message and will not be accepted. However you will

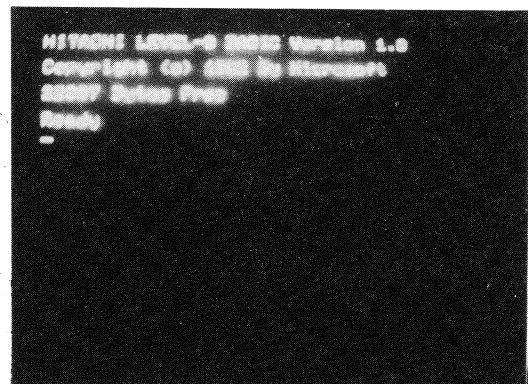


Photo 3.1 Screen Display with Mode Switch at 0 when Power Turned ON

Table 3.2 Screen Mode Occupancy

	normal mode	high res.mode
40 ch	1K bytes	8K bytes
80 ch	2K bytes	16K bytes

Table 3.3 Usable Memory

	normal mode	high res.mode
40ch	29546 bytes	22378 bytes
80ch	28522 bytes	14186 bytes

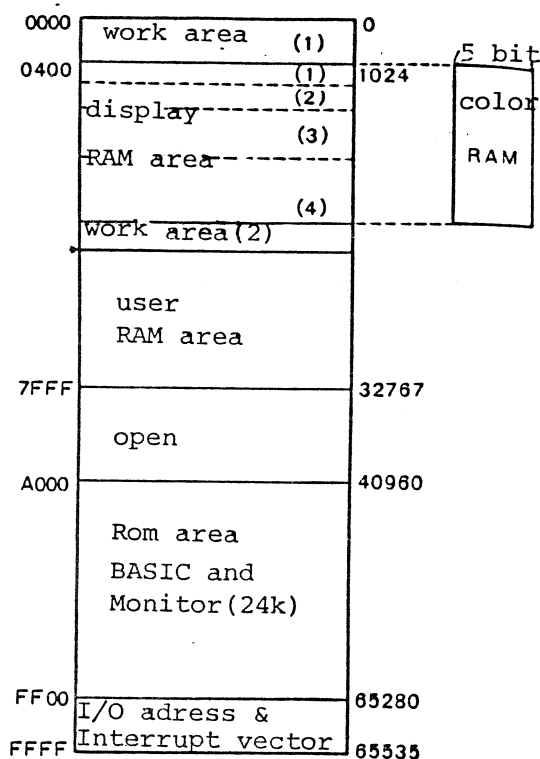
be able to set it at 80-ch. normal mode. These mode relationships are shown in Diag. 3.5. It is thus important to understand which screen mode is set when you turn the power on.

The power is on, you have confirmed the mode and we can start to input the program.

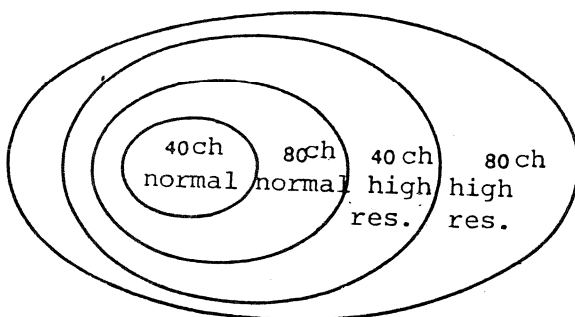
First we should discuss some useful functions the Level-3 has for inputting the program.

* The Return Key

When you press the general input keys, the characters corresponding to those keys appear on the screen. However this does not mean the characters are memorised in the computer. Hitting the RETURN key is necessary to read the character on the screen into the computer. If there is a number at the head of the line, it is regarded as a character string program and stored in the memory. If no number is written at the head, the computer understands it as direct mode and starts immediate execution.



Diag. 3.4 Memory Map



Diag. 3.5 Relationship Between the Screen Setting and the Display Mode

* Screen Editor

Above the TEN keys on the right hand side of the keyboard we have four cursor control keys:



When one of these keys is pressed, the cursor moves in the direction of the arrow. This is rather useful when constructing the program.

Say you find a mistake after inputting the program. For example, you mistyped PRONT for PRINT. Move the cursor using the appropriate cursor control keys and position the cursor right on the O.

```
- PRONT -
  ↑
  cursor
```

Then press the **I** key and the O changes to I and the correct spelling of PRINT appears on the screen. If you press **RETURN** next, the newly corrected line enters the memory replacing the old mistaken program.

The above example is only for one character but what do we do if more than one character needs to be corrected?

Input PRINT incorrectly as PRIT. First, move the cursor to the next character after I to insert N -

```
- PRIT -
  ↑
  cursor
```

Then press **SHIFT** and **INS**
DEL.

The characters from the cursor position on are moved one space to the right and leave that space open.

- PRI T -
 ↑

Press **N** and PRIT becomes PRINT. Then press **RETURN** to store the corrected line in the memory. If you need to insert several characters, then operate **SHIFT** + **INS** the number of times required to leave the necessary spaces. The above function is called INSERT.

The erase function is the exact reverse of the above. For example -

- PRRINT -

This example has an additional R in the word. Move the cursor to the character after the one you want to erase.

- PRRINT -
 ↑
 cursor

Press **INS** and the R next to the P is deleted. The rest of the word is brought up getting rid of the space left by the deleted R.

- PRINT -
 ↑
 cursor

If **RETURN** is pressed, the correction is complete.

The **INS** is also used to delete the incorrect characters.

An example -

10 PRINY
 ↑
 cursor

10 PRIN
 ↑
 cursor

10 PRINT
 ↑
 cursor

As we have seen here, editing of the program on the screen is performed using cursor movement keys and the INS
DEL key. Do not forget to input RETURN to input to memory or all that work will have been for nought.

The cursor key can also be used with SHIFT and, in this case, the cursor moves to the head of each statement or parameter.

100 INPUT J : PRINT "DATA" ; J : IF J > 0 THEN 20

SHIFT + →

100 INPUT J : PRINT "DATA" ; J : IF J > 0 THEN 20

SHIFT + ←

As Level-3 keys have a repeat function after a certain time, you can keep the cursor moving continuously until it comes to the place you want to amend by using SHIFT + → , ←

CHAPTER FOURLEVEL THREE BASIC

The BASIC in the Level-3 has incorporated a lot of previously unavailable new concepts and functions. To be able to make full use of these functions, you will need to understand them well. We recommend that you read the MB-6890 Level-3 Basic Manual but please use the Level-3 to confirm what you read.

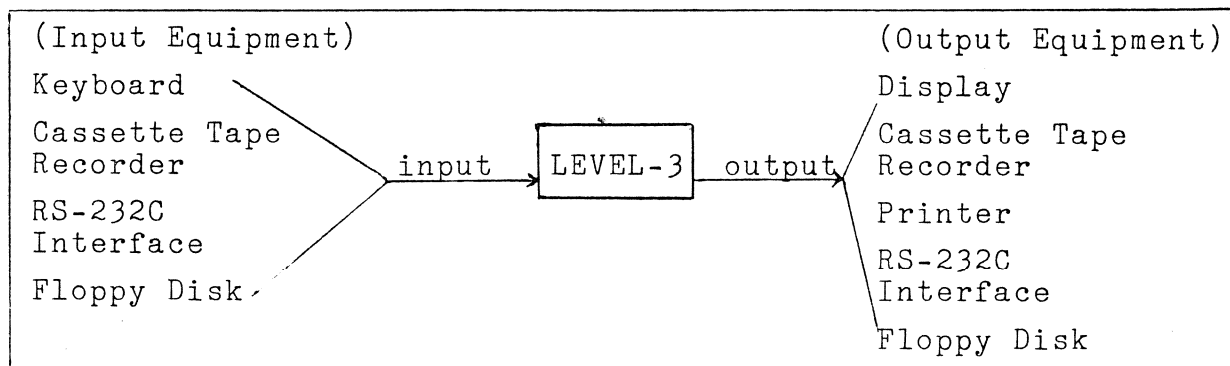
1. Level-3 BASIC Fundamental Knowledge* File Descriptor

This command is to use with peripherals during the execution of the program. Normally these descriptors are written as an I/O device name inside a set of quotes (").

STATEMENT "DEVICE NAME OPTION"

STATEMENT "DEVICE NAME FILE NAME"

↑
file descriptor



Diag. 4.1 Level-3 I/O Devices

Table 4.2 I/O Device Device Name

I/O Device	Device Name	Use
Keyboard	KYBD:	Input
Display	SCRN:	Output
Printer	LPTØ:	Output
RS-232C Interface	COMØ:	Input/Output
Cassette Tape Recorder	CASØ:	Input/Output
Floppy Disk	Ø: -3:	Input/Output

For example -

OPEN "O", , "LPTØ:" ← (file descriptor)

OPEN "I", #1, "CASØ: DATA"
 ↑ ↑
 (file descriptor) (file name)

With the same OPEN command, we can select the peripheral easily by using this file descriptor.

This file descriptor can replace any constant, variable or expression. So the one program can be used with many peripherals.

```
OPEN  "O",  #1, A$
{      ↑
      "LPTØ:" "SCRN:" "CASØ:"
CLOSE
```

Sometimes the device name, option or file name is omitted but a detailed explanation follows in a later section. For the moment, just absorb what a file descriptor is.

For information on character constants, numerical constants, variables and such please read the relevant section in your MB-6890 Level-3 Basic Manual.

2. Mastering the Commands

We will talk here only about the commands themselves. An explanation of the statements is included in a later section.

In general, commands should not be written into the program but sometimes we write some of them to execute instructions rather like statements.

* LIST - Display the program in the memory

LIST is the command to output the program to the screen. It is used to check the program in the memory and correct it if called for. Input LIST and all the program, starting from the first line number, appears on the screen. A long program cannot be displayed on the screen all at once so it is scrolled from the first line to the last. However, we are not able to spend a lot of time on a particular part of the program. To do this, we add a line number option to the LIST command.

1. LIST n ... display only line number n
2. LISTn,)
 LIST n-) ... display line numbers from n
 onwards
3. LISTm, n)
 LISTm-n) ... display line numbers m to n
4. LIST, n)
 LIST-n) ... display up to line number n

If no line number had been designated by LIST command in the above examples, the following results would have been obtained:

1. No display and input wait.
2. Start displaying from the next largest line number designated by the command. If no program exists after the line number n, there would be no display and input wait.
3. The first m would have the same operation as in 2. and for the next n, the output would stop one line after n.

We use this command so often we can use the abbreviation, L. instead. This does the same work as LIST. Also, the program function key, `[PF4]`, has the definition of LIST + `[RETURN]`. So, by simply pressing `[PF4]` we can accomplish the same operation. However, note that we cannot use `[PF4]` to designate the line number. On these occasions, use L.

LIST has another interesting function. With the abbreviation we use (.) but in LISTn, the line number n can be substituted by this (..).

Normally, we use LIST + line number but if only LIST. is used, the following line numbers described in 1. to 4. are set. Abbreviating this further, L.. can be used to list a designated line number.

1. The line number of the error.
2. The STOP or END line number when the program has been stopped by a STOP and/or END statement.
3. The line number the program has stopped at when stopped by `[BREAK]` or `[CTRL] + [C]`.
4. The last line number the LIST has been executed to.

We consider how to list the program using `[CTRL]`.

When the error message appears on the screen, it is easy to correct the error by listing the designated line number on

the screen. However, if you cannot obtain the expected result even when the program is grammatically correct, we must check the whole program. At times like these, using `CTRL + S` is a convenient method. Execute LIST and when you come to a place you would like to stop at, hold down `CTRL` and press `S`. The scrolling stops and you can check this part of the program carefully. To re-start the scrolling, press any key other than `CTRL` or `S`. If you want to return to the command input waiting state, press `BREAK` or `CTRL + C`. The screen displays ready and the cursor and you are in the command input state.

Next, we will consider outputting the list to other peripherals.

Firstly, how to output the program list with the printer. You will remember the LPT0: file descriptor we mentioned earlier.

```
LIST "LPT0:"
```

When you execute the above program, the program in the memory is output by the printer. Of course, you can output particular lines by giving the line number like this-

```
LIST "LPT0:" ,50
LIST "LPT0:" ,100-600
L. "LPT0:", ,200
```

When you output the list through RS-232C -

```
LIST "COM0:"
LIST "COM0:" , 500
```

When you output the list to the screen -

```
LIST "SCRN:"
LIST "SCRN:" ,100-300-
```

* RUN - Program execution

The command to start program execution is RUN. When you input RUN, the program is executed in order starting from the head.

If you want to execute the program from a designated line number -

RUN n

and add the designated line number after RUN. For example if you execute the program from line number 250, you should input -

RUN 250

Be careful after stopping the program with **BREAK**. If you input RUN or RUN n after this and re-start execution, all the previous data will be erased.

The abbreviation of RUN, R. can also be used -

RUN → R. RUN 250 → R. 250

* CONT - Re-start the program

When the program has been stopped with **BREAK**, STOP or END and you then input CONT (CONTInue), program execution re-starts from the line after the line it has stopped at. In this case, the data and variables included in the program up to the point at which it had been stopped are kept.

In contrast to this, we cannot operate CONT if an error occurs and the program has stopped, or, if we stop the program to make a correction. If we try to enter CONT we get a 'Can't continue' error message.

The abbreviation of CONT is C..

These three commands, LIST, RUN and CONT are so often used that they are defined in the programmable function keys, namely -

LIST + <u>RETURN</u>	=	<u>PF4</u>
RUN + <u>RETURN</u>	=	<u>PF5</u>
LIST "LPT0:" + <u>RETURN</u>	=	<u>PF4</u> + <u>SHIFT</u>
CONT + <u>RETURN</u>	=	<u>PF5</u> + <u>SHIFT</u>

Both the command and RETURN code have been defined in the function keys. Therefore, we cannot use them when we have line numbers. For this we must use full spelling or the relevant abbreviation.

* NEW - Erase the program

To input a new program, it is necessary to erase the old program. If the old program is not erased, it will mix with the new program and, as a result, the program will not work properly. When you input a new program, we use the NEW command to erase the old ones. Note that executing the NEW command does not alter the program area or the screen state. There is no abbreviation for NEW.

3. Making Programming Easier

* AUTO - Automatic line number generator

As you know, we must write a line number at the beginning of each line in BASIC programs. The Level-3 has an AUTO command which generates a line number in intervals of 10.

When you input AUTO, line numbers from 10 on are output in increments of 10. If you want to start from line number 100 we input -

AUTO 100

and line numbers are automatically generated in increments of 10 from 100 on.

To generate the line number from 100 at intervals of 20 we input -

AUTO 100, 20

and the numbers are output 100, 120, 140

In summary,

AUTO m, n - m is the first line to be generated
 - n is the increment of each line no.

Either parameter of the AUTO can be omitted as demonstrated here -

AUTO ➔ AUTO 10, 10
AUTO 20 ➔ AUTO 20, 10
AUTO, 30 ➔ AUTO 10, 30

All the numbers generated by AUTO must be positive integers. m must lie in the range of 0 to 63999 and n between 1 to 63999 integers.

An advantage of the AUTO is that the command execution will discontinue if a line number generated is the same as a line number in an existing program. This prevents needed data in the memory being erased inadvertently.

Let's now cancel AUTO. There are several ways to do this. Press one of the following keys -

CTRL + C
CTRL + D
BREAK
RETURN

Note that when more than two lines of a program are written on the screen, the program is input to the memory but the

AUTO is disabled.

* RENUM - Renumber the line number

When you are making a program, sometimes the line number intervals become irregular. The Level-3 has a RENUM command to RENUMber line numbers in the same intervals.

```

RENUM l, m, n
      ↑   ↑   ↑
      |   |   |
      |   |   | line number increment
      |   |   | present program line number
      |   |   | intended renumbered line number

```

For example -

RENUM 100, 30, 10

This means the line numbers from 30 on in the program presently stored in the memory are to be renumbered 100 and on in increments of 10. Thus -

10 A=B		10 A=B
15 C=J+10		15 C=J+10
20 A=A+1		20 A=A+1
30 B=SIN(A)	RENUM 100, 30, 10	100 B=SIN(A)
35 ..	→	110 ..
36 ..		120 ..
40 ..		130 ..

When we change the head line number, we must also update the line numbers showing the place we skip to in GOTO, IF and GOSUB statements. This RENUM statement automatically changes these designated line numbers too. For example -

30 ..		100 ..
40 ..	RENUM 100	110 ..
45 ..	→	120 ..
50 GOTO 40		130 GOTO 110

If the designated line number does not exist in the program, the 'undefined line m in n' message appears on the screen.

m is the line number which cannot be found in the program; n is the designated line number. In this case, the designated line numbers in error remain as they are without being changed.

The parameters of RENUM may be omitted -

RENUM → set the head new line number at 10 with increments of 10

RENUM 100 → set the new head line number at 100 with increments of 10

RENUM ,5 → set line number 5 to 10 with increments of 10

RENUM ,,5 → set the head line number at 10 with increments of 5

RENUM 1,,1 → set the head line number at 1 with increments of 1

You can see that RENUM is quite useful for re-ordering line numbers of the input program to make them easier to follow.

* DELETE - To delete a part of a program

When we make a program normally there are lines which we later realise are unnecessary. If there are only one or two lines, input the line numbers, press **RETURN** and they will be erased. However, this method would be rather bothersome if we had to erase more than ten lines. For this, we use the DELETE command.

DELETE m, n	m = the first line number to be
or	deleted
m - n	n = the last line number to be
	deleted

In this way, all the lines from m to n would be deleted. Either (,) or (-) may be used.

Use this command with care as you may delete needed lines by mistake. Not all of the DELETE parameters may be deleted -

DELETE 105-290 → Delete line numbers from 105 to 290

DELETE 500, → Delete all line numbers from 500

DELETE ,200 → Delete all line numbers from the head to 200

DELETE 400 → Delete 400 only

DELETE . → Delete the line the BASIC pointer shows

* EDIT - Edit the program

In addition to the screen editing function the Level-3 has an EDIT command to correct the program.

When you input EDIT n, the line number n appears on the screen and we are ready to correct that line. For example, if we input -

EDIT 100 we get

```
EDIT 100
100 PRINT "LEVL3"
    ↑
  cursor
```

The cursor is brought to the head of statement 100. We can then move the cursor to the place we want to correct.

```
100 PRINT"LEVEL3"
```

The line number for the EDIT must exist in the program. If not, the 'Undefined Line Number' error message appears on the screen. You can also correct a line by using the pointer

(.) in BASIC thus -

RUN

Syntax Error In 30

Ready

EDIT.

30 FOT I=0 TO 30

↑ this should be R
cursor

* TRON, TROFF - To check the flow of the program

Sometimes, though the program itself is not grammatically wrong, in theory it may be utter nonsense. If this happens, you will be in a quandary about how to debug it. When having problems with your program execute the trace mode. If you input trace mode, all the program line numbers as they are executed, appear on the screen. With this, we know for sure the lines executed and can tell at a glance the offending lines.

To put the BASIC into trace mode use the TRON command. Then press RUN. The screen displays the line numbers in [] These are the programs which are executed.

TRON

Ready

RUN

(10)(20)(30)(40)(50)(60)(70)(80)

To discontinue this execution, press **CTRL** + **S**.

When we return to the input level, press one of the following - **BREAK**, **CTRL** + **C**, **CTRL** + **D**. Please note that the trace mode is still continuing.

To release the trace mode use the TROFF command. Input

TROFF

and the trace mode is disabled.

We can also write TRON and TROFF with line numbers -

```
5  
500 TRON
```

```
5  
600 TROFF
```

In the program execution, when TRON is executed, BASIC goes into the trace mode and this mode is released by TROFF. Executing TRON for trace mode and TROFF for non-trace mode will not cause an error. So, it does not matter where in the program these two commands are inserted. They will have no effect on the program.

CHAPTER FIVE

STATEMENTS

1. LET Statement

LET was used in the early stages of BASIC for arithmetical formulae. This LET is now omitted in present programs -

10 LET A=B → 10 A=B

A few points on the LET statement to consider -

- (1) The variable should be written on the left hand side and the expression on the right.
- (2) The variables on the right hand side must be defined in a previous line. Undefined variables are counted as 0
- (3) The variables on either side must be the same in form.

These following examples demonstrate correctly formed LET statements -

```
C = A/B
A$ = B$+"LEVEL 3"
B = SIN(Y)+COS(Z)*3.14159
A = B > 10
X = ((A >= 30) AND (B < 10)) * (-68)
```

Please refer to the MB-6890 Level-3 Basic Manual for expression operation and logical operators.

2. Input/Output Instructions

Input/output instructions are for inputting data from the keyboard and outputting the results of the execution to the screen. There are several statements.

* INPUT - Basic Statement of Input

This is the most frequently used statement in the input/output instructions. You must prepare variables for data input. So always put the variable after INPUT. Both arithmetic and character variables are acceptable.

Let's try out the INPUT function -

```
10 INPUT A,B
20 PRINT A*B
```

When you execute this program, the screen shows ? and goes to input wait -

RUN

?_

Input 6, 9 for the variables A and B and the following output appears on the screen -

```
?6,9
54
```

Note the input data is separated by (,).

Let's try an example using character variables next -

```
10 INPUT A$,B$
20 PRINT A$,B$
RUN
?HITACHI,LEVEL3
HITACHI      LEVEL3
```

We can input almost all the characters but we should use (") if we input (,) (comma) and (:)(colon) since they carry a separation symbol meaning.

In the input waiting time after RUN when only ? appears on

the screen, it is sometimes difficult to know what we should input. It would be very useful if there were a message to instruct us on what we should input. And, for this, we have a character string output function.

Enclose the character string you want to output in ("), input (;) and write the variable. Let's try our previous example using this method -

```
10 INPUT "A,B";A,B
20 PRINT A*B
RUN
A,B?6,9
54
```

What will happen if we input non-numerical data when set for numerical data with the INPUT statement?

```
10 INPUT A
20 PRINT A
RUN
?H
?Redo From Start
?_
```

'?Redo From Start' means your input is wrong so re-input the correct message all over again. There are also times when the number of input data is different from that set by INPUT. The program does not stop so re-input the correct data.

* LINE INPUT - Any characters are input

All characters including (,) and (:) which cannot be input by INPUT can be input with the LINE INPUT statement. This statement can input character strings to 255 characters, and like INPUT, also has the function of outputting character strings.

* INPUT WAIT - Input with a time limit

This is the statement for input with a limit on waiting time. The BASIC waits for input for a number of seconds designated by the wait time. If no input is made within that time limit, the control moves to the line number designated. Numbers between 1 and 255 may be used for this waiting time. Apart from the waiting time function, this command is the same as the INPUT statement.

An example -

```
10 INPUTWAIT 40;5,"ENT";A
               ↑   ↑
               |   waiting time
               |
               designated line if no input is
               made during waiting time
```

With this program, ENT is displayed and sets a 5-second waiting time. If no input occurs within five seconds, the control moves to line number 40.

In this INPUT WAIT, you must write the designated line number even if it only the next line -

```
10 INPUTWAIT 40;5,"ENT";A$
20 PRINT"COMPLETE ";A$
30 END
40 PRINT"INCOMPLETE"
50 END
```

Let's execute the above program -

```
RUN
ENT?_
```

The output appears as shown above and if we have input within the five seconds, COMPLETE is displayed. If no input is made within the five seconds, the control moves to line 40, displays INCOMPLETE and the input wait command ends.

To input the data, you need to press RETURN after pressing the character keys. Therefore, the waiting time includes this RETURN input time as well.

Let's now mention the output statements which are PRINT and PRINT USING.

* PRINT - Basic statement for output

All output to the screen is executed with PRINT. Let's check the functions of PRINT -

```
PRINT "ABC"  
ABC
```

```
PRINT 123  
123
```

```
PRINT 60+38/2+5  
84
```

We can see that character strings, numerical expressions and calculation results can be output. There is also the advantage that calculations are implemented with multiplication and division taking precedence - unlike the case of a calculator.

This command is used so frequently that we have an abbreviation ? for it -

```
?"ABC"  
ABC
```

PRINT is also used to output the contents of the variables - both numeric and character string variables .

```
PRINT K  
PRINT A$
```

There are no restrictions on the sequence of expressions, numerics and variables.

```
PRINT A$;
PRINT "NO.", K
PRINT N$; "DATA", F, K+J
PRINT L+50, I
.
.
.
```

Separators such as (,) and (;) as in the above examples should be used when the characters and numerical expressions are written in the same PRINT statement. For example -

```
PRINT "LEVEL3","BASIC"
LEVEL3      BASIC
```

When there is no separator after a character expression, the latter (") may be omitted.

The display area set by PRINT is divided as in Fig. 5.1. When (,) is used as the separator the display starts from the head of each area.

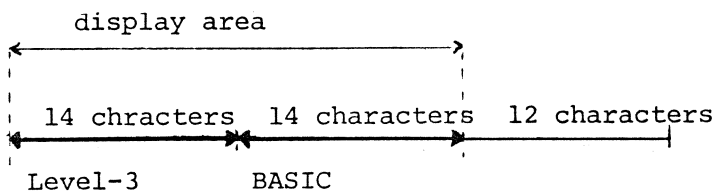


Fig. 5.1 40 ch. Mode Display Area

What about when we input this?

```
PRINT , "LEVEL3","BASIC"
```

In this case, (Level3) is moved the number of places to the (,) and output as follows -

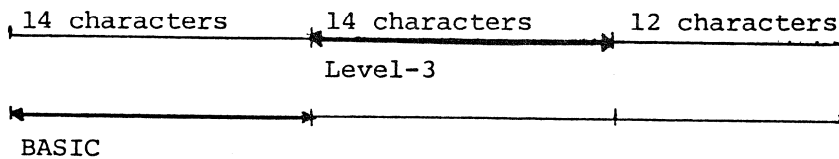


Fig. 5.2 Input after Moving the Head

(BASIC) comes up on the next line. With the (,) separator, the last twelve characters cannot be used in 40-ch. mode and likewise, the last ten characters in 80-character mode may not be. If the words run over into this area they are output on the next line.

If the characters to be output are to start from the second display area and have more than 26 characters, the printout will start on the next line instead. This is different to the case in which BASIC outputs on a following line the part that was not able to fit on the one line.

If you use the (;) separator, disregard the display area and display the characters continuously. The (;) may be omitted when delimiting numerics.

If (;) or (,) comes at the end of a PRINT statement, it joins the next PRINT statement -

10 PRINT"BASIC";	10 PRINT"BASIC"
20 PRINT"MASTER"	20 PRINT"MASTER"
↓ RUN	↓ RUN
BASICMASTER	BASIC
	MASTER

If PRINT is used alone, we are moved to the next line -

10 PRINT"BASIC";	10 PRINT"BASIC"
20 PRINT	20 PRINT
30 PRINT"MASTER"	30 PRINT"MASTER"

↓ RUN

BASIC
MASTER

↓ RUN

BASIC
MASTER

PRINT USING - Format control output

There are often times when you want some output to have the same number of decimal places or when you perhaps want to punctuate numbers with (,) every three units. It is rather difficult to do with just the PRINT statement and requires some degree of program technique. If you are confident, try it!

For times like these, we have the PRINT USING command which we can control the format output with.

PRINT USING "format expression";expression string

(1) Format Control on Character Variables and Character Strings - ! &

By using ! only the first letter of a given character variable or character string is output.

Let's assume that we have input A\$ for BASIC, B\$ for MASTER and C\$ for LEVEL 3 -

PRINT USING"!";A\$;B\$;C\$

↓ RUN

BML

Look at the next one -

PRINT USING"& &";A\$;B\$;C\$

↓ RUN

BASIC MASTER LEVEL 3

& is used to display only a designated number of characters from the head of the character string shown between a pair of "&"s as in the previous example. Both the symbol & and the spaces are taken into account so that $(2 + N)$ characters are reserved, N being the number of blanks.

In the above case, an area of ten characters has been reserved, so that all the characters stored in the variables A\$, B\$ and C\$ have been output. If the character variable has more than ten characters, only the first ten characters of the variable are output from the left and the rest is ignored.

```
PRINT USING"&  &";A$;B$;C$
↓      RUN
BASIMASTLEVE
```

We can also write other characters with the characters designated by ! and (& &) -

```
PRINT USING"X ! ";A$;B$;C$
↓      RUN
X B X M X L
```

```
PRINT USING"&      & DX ";A$;B$;C$
↓      RUN
BASIC      DX MASTER      DX LEVEL3      DX
```

```
PRINT USING"! &      & !";A$;B$;C$
↓      RUN
B MAST L
```

(2) Numeric display format control - # and . # and .

and .

(.) is the decimal point and (#) is the number column to be displayed. Using this, we can output multiple numbers arranged in a particular decimal point display. We can also

select to output a certain number of decimal places. When the decimal part of a number is bigger than the number of places designated, the relevant part is rounded. For example -

```
A=3.14:B=159.36:C=207.3
```

```
PRINT USING"###.##":A;B;C
```

```
↓ RUN
```

```
3.14 159.36 207.30
```

```
A=3.1415:B=159.367:C=207.309
```

```
PRINT USING"###.##";A;B;C
```

```
↓ RUN
```

```
3.14 159.37 207.31
```

% is displayed at the head of the numeral to indicate when the integer part of the number exceeds the designated number of columns -

```
A=1918.362
```

```
PRINT USING"###.##";A
```

```
↓ RUN
```

```
%1981.36
```

If the variable is a numeral with decimal places and in accordance with the form prescribed no decimal point is to be used, only the integer part is displayed. Here also we round off after the decimal place.

```
A=1918.382
```

```
PRINT USING"####";A
```

```
↓ RUN
```

```
1918
```

```
A=1918.883
```

```
PRINT USING"####";A
```

```
↓ RUN
```

```
1919
```

and ,

In everyday office calculations, (,) is used to display numbers marked off in units of three.

```
PRINT USING"###,###,###";A$      (A# = 1 2 3 4 5 6 7 8 9)
```

↓ RUN

123,456,789

With the PRINT statement, the output is from the left but with the above program, (,) is inserted every three places from the right. Look at the next example -

```
PRINT USING"#####,";A      (A = 1 2 3 4 5 6)
```

↓ RUN

123,456

Likewise the output is marked off in units of three. This means output numerals may be delimited with (,) only every three digits.

As # designates the columns, firstly, it reserves the number of spaces and then it punctuates the number in units of three where (,) appears. Therefore, (,) may be placed anywhere except at the head of the character. Take note of the following example however. We input a 9-digit number for a double precision variable A# -

```
A#=123456789
```

```
PRINT USING"###,###,###";A$
```

↓ RUN

123,456,789

```
PRINT USING"#####,";A$
```

↓ RUN

%123,456,789

With the usual designation, if you input to delimit every three digits, the output will be just that. However if the (,) is omitted, the column number overflows and the + or - symbol area is not reserved and the symbol % is added to the head of the numeral when displayed. If, in this case, you

write the output # as maximum column number + 1, the column overflow will not occur.

Normally, the symbol area is also included in the # number, so if you designate only the number's maximum column number and a negative number is output, it will overflow and % will be output. When only positive numbers are used, there is no such problem since the symbol is disregarded.

+ and -

Sometimes, outputting a symbol causes the numeric format to become disordered. In this case, it is clearer if you add the symbol on after the number. The format of this command is -

```
PRINT USING "###, ###, ###+ or":A#  
-
```

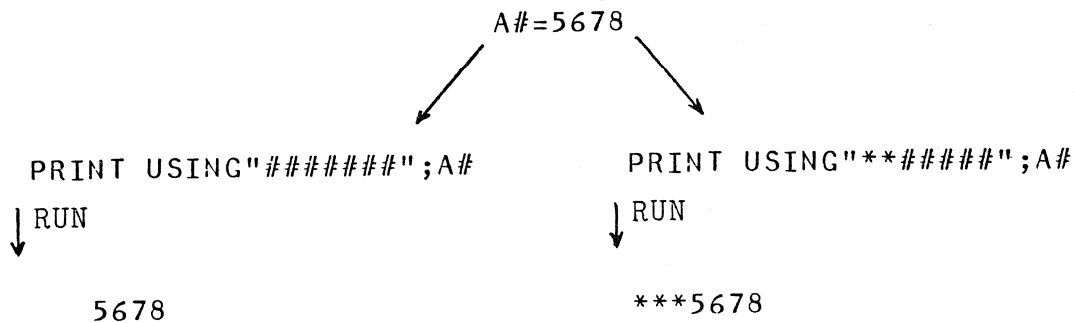
When "+" is used, a plus sign is output with a positive numeral and a minus sign is output with a negative one.

When "-" is used, a positive numeral is output with a blank (the + is omitted) and a minus sign is output with a negative numeral.

A\$=123456:B\$=- 32768	
↙	↘
<pre>PRINT USING"###,###+";A#;B# ↓ RUN 123,456+ 32,768-</pre>	<pre>PRINT USING"###,###- ";A#;B# ↓ RUN 123,456 32,768-</pre>

**

Up until now, if the number of columns in the number has been fewer than the designated column area, blanks have been output in the spare places. But with the "***" designation, the empty places are filled with *s. ** reserves a 2-column area the same as two #s.



If only one * is used, it is regarded as a simple character only -

```

A#=5678
PRINT USING"*#####";A#
↓ RUN
* 5678

```

* ^ ^ ^ ^ EXPONENT DISPLAY DESIGNATIONS

For the effective column number, double precision constants are stored in a degree of precision of 16 digits and single precision constants to a precision of 6 digits. When you output a numerical variable of more digits with a PRINT statement, we get the following display -

```

10 A=1580000
20 PRINT A
RUN
1.58E+06

```

The format is -

```
PRINT USING "#####^/^/^";A
```

The designation of ^ must always be as a group of four placed after #. The expression of the exponent part is $E^{\pm nn}$, for single precision and $D^{\pm nn}$ for double precision. If you use this, you can display the exponent part of the numeral completely ordered on the screen.

The effective digit number is decided by the number of #s.

* Application of PRINT USING

PRINT USING is used especially for data base management. We have already mentioned how to use this statement and we will now consider its applications. Look at this program -

```

10 SCREEN0
20 WIDTH 80
30 DIM A(4)
40 L$=STRING$(50,"-");S$=SPACE$(6)
50 ' print title
60 PRINT TAB(10);"## Print using sample chart ##"
70 PRINT L$
80 PRINT "No.";SPC(5);"data1";S$;"x 0.8";S$;"x 0.5";SPC(5);"x 0.25"
90 PRINT L$
100 ' data make
110 FOR I=0 TO 15
120 D=INT(RND(1)*100000);A(1)=A(1)+D
130 D1=D*0.8;A(2)=A(2)+D1
140 D2=D*0.5;A(3)=A(3)+D2
150 D3=D*0.25;A(4)=A(4)+D3
160 PRINT USING"###";I;
170 PRINT USING"    ###,###";D;D1;D2;D3
180 NEXT
190 ' print total
200 PRINT L$
210 PRINT"TOTAL:";
215 FOR I=1 TO 4
220 PRINT USING"  ##,###,###";A(I);
230 NEXT

```

When you execute this program, we get a table as in Photo 5.2 output to the screen.

You must prepare the data for this statement but random numbers are generated in this program and used as data.

Line number 120 generates the random numbers, and line numbers 120-150 construct four sets of data from these random numbers.

Line 180 is the FOR-NEXT loop to display the 15 line x 4 row data format.

No.	DATA	x 0.8	x 0.5	x 0.25
0	29,786	23,829	14,893	7,447
1	28,789	23,031	14,394	7,197
2	27,792	22,233	13,895	6,947
3	26,795	21,435	13,396	6,697
4	25,798	20,637	12,897	6,447
5	24,801	19,839	12,398	6,197
6	23,804	19,041	11,899	5,947
7	22,807	18,243	11,400	5,697
8	21,810	17,445	10,901	5,447
9	20,813	16,647	10,402	5,197
10	19,816	15,849	9,903	4,947
11	18,819	15,051	9,404	4,697
12	17,822	14,253	8,905	4,447
13	16,825	13,455	8,406	4,197
14	15,828	12,657	7,907	3,947
15	14,831	11,859	7,408	3,697
TOTAL	712,066	569,653	356,080	178,047

Photo 5.2 Table Using
PRINT USING (1)

After this loop, the aggregate variables A(1), A(2), A(3) and A(4) are displayed.

Line 170 is the PRINT USING statement to display the data of each variable D, D1, D2, D3. The format of each piece of data is four spaces at the head of the numeral which has an area of six digits reserved, (,) to be inserted at the third digit of the numeral.

When each piece of data has the same format, each one is designated with only ; delimiting the variables.

Let's change this program a little to enable us to input the data from the keyboard. First add the following lines -

```
25 CONSOLE 0,25
105 CONSOLE 5,14
115 LOCATE 0,23:PRINT CHR$(26)
121 LINEINPUT D$:IF D$="" THEN 121
122 IF D$="*" THEN 230
123 IF D$="/" THEN RUN
125 D=VAL(D$):A(1)=A(1)+D
155 Y=I:IF I>13 THEN Y=14
225 NEXT:NEXT
230 CONSOLE 0,25
```

Delete line number 180 and change line nos. 110, 120, 160, 200 and 210 as shown below:

```
110 FOR I=1 TO 100
120 LOCATE 0,23:PRINT "No. ";I;": ";
160 LOCATE 0,4+Y:PRINT USING "###";
I;
(delete Line 180)
200 LOCATE 0,19:PRINT L$
210 LOCATE 0,20:PRINT "TOTAL:";
```

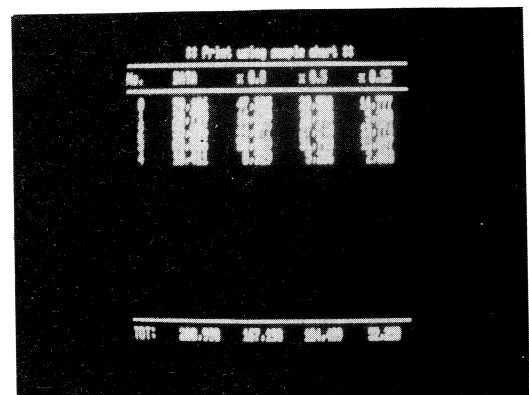


Photo 5.3 PRINT USING Table(2)

Execute this revised program. After displaying the title and No. n on the bottom line of the screen, it goes to the data input waiting condition. See Photo 5.3.

Input the data next; and the original data together with the

new values obtained by multiplying by 0.8, 0.5 and 0.25 appear on the screen.

This program starts scrolling after inputting data No. 14 and a total of 100 pieces of data can be input. The title and the items remain the same so that continuous data input is possible.

* READ, DATA, RESTORE

We have already talked about INPUT. We will now explain the READ statement, another data input statement.

The INPUT statement inputs data when the program is executing but this can be inconvenient when we use data arrived at during execution. For this situation we can use the READ statement.

READ operates jointly with the DATA statement (or RESTORE) by writing a variable after READ and the corresponding data after DATA. DATA can be placed at any point in the program. Writing it immediately after the READ line or on the last line is probably easiest.

```
10 READ A,B,C
20 V=A*B*C
30 PRINT A;B;C:V
40 DATA 30,25,86
```

In this example A=30, B=25 and C=86.

```
10 FOR I=1 TO 5
20 READ A
30 PRINT A;
40 NEXT
50 DATA 5,8,9,1,6
```

The data can be written in many lines as long as they have corresponding variables in the READ statement.

Normally the constants in the DATA statement can only be used once. But, by using RESTORE, it becomes possible to re-read the same DATA statement -

```
10 FOR I=1 TO 5
20 READ A
30 PRINT A;
40 NEXT
50 PRINT
60 RESTORE
70 GOTO 10
80 DATA 5,8,9,1,6
```

In this example, the RESTORE statement is on line 60. Without it an 'Out Of Data' (insufficient data in DATA statement) error message appears.

With the extended form of the statement, RESTORE "line number", the DATA after the line number specified can be used many times.

To read characters with a READ statement the character string is enclosed in quotes (") the same as with a PRINT statement -

```
10 FOR I=1 TO 4
20 READ A$
30 PRINT A$
40 NEXT
50 DATA "HITACHI", "LEVEL3", "COMPUTER", "MB- 6890"
```

This is the same as reading numerical values. We can omit (") from the data of this character string but if we do so the (,) will not be read as a character.

Next, we will introduce an interesting use of the READ statement -

```
10 FOR I=0 TO 10
20 READ A$
30 DA=VAL("&H"+A$)
40 POKE &H5000+I,DA
50 NEXT
60 DATA 0F,CA,03,8A,BF,C3,5B,CD,F0,A3,DA
```

For storing this hexadecimal data in the memory directly or inputting a machine language program to the memory, there is a method whereby we write the hexadecimal data and machine code in the DATA statement and read them with the READ statement.

In this program, the data is once stored into variable A\$ as characters and then converted to a numerical value in line number 30.

VAL is the function to convert characters to numerical values. The argument is &H (hexadecimal number) which is placed at the head of the data which is read by the READ statement. The converted numerical data is set at variable DA. We can then write this data directly to memory with a POKE statement.

This method is not suitable for large-scale data writing because it would take too long but it is most effective for short programs.

The advantage of this method lies in the fact that the commands are in the BASIC program. We can thus keep them as is and perform corrections easily by displaying the machine code section with LIST.

When octal data is written, &O is used instead of &H. However it is not used that often as hexadecimal are more common with microcomputers.

3. Other Important Statements

FOR I = a TO b STEP c

(1) When $a < b$, $c > 0$ -

The loop variable I is set first at a and increases up to b in increments of c.

(2) When $a > b$, $c < 0$ -

The value of I becomes smaller with each execution.

(3) Others - $a < b$, $c < 0$ or $a > b$, $c > 0$ -

There is no grammatical error but the program will not be executed.

Workable Programs:

```
10 FOR J=1 TO 10
20 PRINT J;
30 NEXT J
```

```
10 FOR J=10 TO 1 STEP -2
20 PRINT J;
30 NEXT J
```

These Programs will not Work:

```
10 FOR J=10 TO 1
20 PRINT J;
30 NEXT J
```

```
10 FOR J=10 TO 1 STEP 2
20 PRINT J;
30 NEXT J
```

NEXT stands for the end of the loop and is typed as NEXT I after the variable name in the FOR part. When this command is executed, the next value is set in the loop variable.

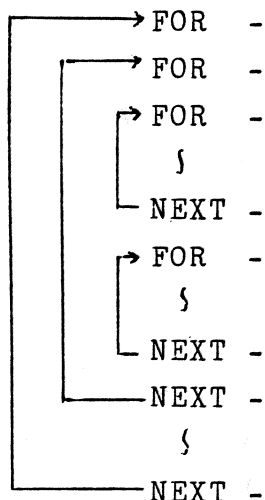
We can use a second FOR-NEXT loop within a FOR-NEXT loop. We call this loop structure nested loops.

```

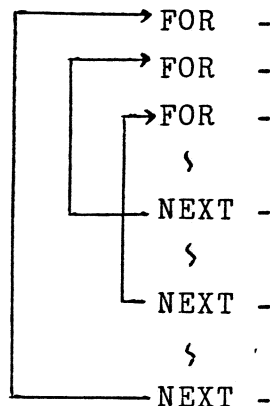
  → FOR J = 1 TO 30
    → FOR I = 2 TO 5
      ↵
    NEXT I
  NEXT J
```

If the loops cross each other you will get an error message.

Correct Loop -



Incorrect Loop -



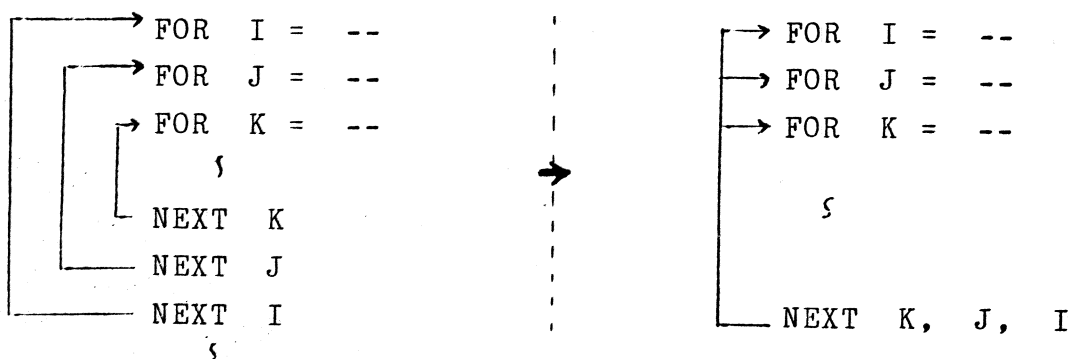
Each loop variable name must be different -

```

10 FOR I=1 TO 10
20 FOR J=1 TO 10
30 PRINT I*J;
40 NEXT
50 PRINT
60 NEXT

```

Several nested loops may end at the same line number. In this case, we can make one NEXT statement represent all of them. When you do this though, be careful to write the loop variables after NEXT in order from the inside loop variable to the outside. If the variable order is wrong, it will mean an error.



The number of FOR-NEXT loop nested is limited by the memory capacity available but is not inconvenient if you keep away

from overcomplicated ones.

* DIM - Array variables

We call such variables as A(1), A(2), A(3), array variables. When we use them in a program, we must define them with a DIM statement such as -

```
DIM A(10)
```

A DIM statement states the array variables. The above example reserves eleven variables, A(0), A(1) ...A(10), in the memory. The number in () is called the subscript and this is what decides the position.

The subscript may be a numerical constant, a variable or a formula but it must be 0 or a positive integer.

```
10 DIM A(10)
20 FOR I=0 TO 10
30 A(I)=I*2
40 NEXT I
50 FOR I=0 TO 10
60 PRINT I;A(I)
70 NEXT I
80 END
```

Level-3 array variables can have many subscripts which means we set a multi-dimensional array. Stating it is the same as when the subscript is only one. For example, we state a two-dimensional array A up to A(8,8) as -

```
DIM A(8,8)
```

A three-dimensional array B up to B(3,3,3) is declared as -

```
DIM (3,3,3)
```

In a two-dimensional array the memory makes a graph as in Fig. 5.4 so that the vertical and horizontal parameters set

one variable.

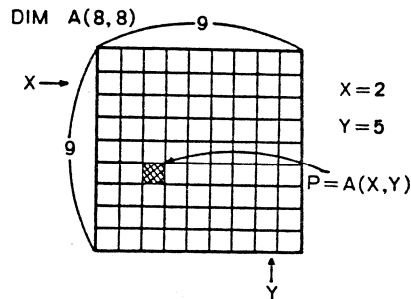


Fig. 5.4 2-dim. setting

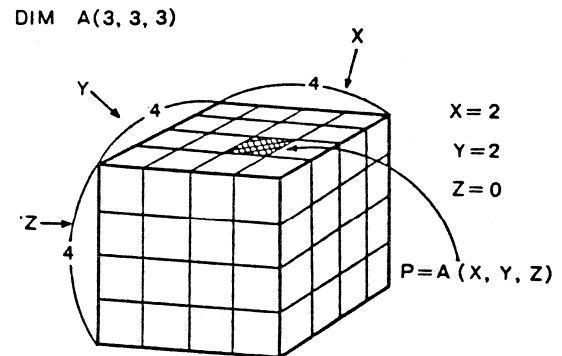


Fig. 5.5 3-dim. setting

If the number of dimensions becomes bigger, the memory required also increases, which may lead to an impossible situation.

A DIM statement can also set up character variables as well as numerical variables -

```
DIM A$(10)
```

Character array variables are used to make tables and to execute the data in the tables.

* IF - THEN - ELSE - program

An example -

```
IF A = 0 THEN 500 ELSE 300
           ↑       ↑
           GOTO 500 GOTO 300
```

Or, when GOTO follows THEN we can delete THEN and write -

```
IF A = 0 GOTO 500
```

The main purpose of this condition format is comparing whether larger or smaller but together with AND or OR we can judge multiple conditions at the same time.


```

IF A > B THEN  --
IF A=B OR C=D THEN  --
IF A > B AND C < D THEN  --

```

* ON-GOTO, ON-GOSUB - Transfer program control by
calculation result

The GOTO and GOSUB statements do not allow the program to be transferred to a designated line number through the result of a calculation. This number should be a decimal integer constant. To change the line number of transfer you can use an IF statement but you can only move one line.

If you use ON-GOTO instead of an IF statement, the transfer can be easily effected -

```

IF A = 1 THEN 320
IF A = 2 THEN 580
IF A = 3 THEN 530
IF A = 4 THEN 220
IF A = 5 THEN 950

```



```

ON A GOTO 320, 580, 530, 220, 950
           ↑      ↑      ↑      ↑      ↑
           A=1    A=2    A=3    A=4    A=5

```

The transfer line number is selected by the variable between ON and GOTO. The variable must be an integer of 1 or more. If the variable is 0 or if it is greater than the count of the line number, the control moves to the next line as if the conditions were not met.

ON-GOSUB is the command to call a subroutine. When control returns from the subroutine, the next line is executed.

```

ON  variable  GOSUB  number, number, ....
      or
      expression

```

ON-GOTO, ON-GOSUB are called respectively calculative GOTO and calculative GOSUB statements and depending on the result, you can select the branch.

* DEF INT/SNG/DBL/STR

There are four types of variables - integer variables, single precision variables, double precision variables and character variables. Each is displayed with its respective attribute character added on the end -

A%, NUMBER%	...	Integer
A!, VALUE!	...	Single precision
A#, B2#	...	Double precision
A\$, CHARACTER\$...	Character

Normally, ! is omitted if the variable is the single precision type. That is to say, such variables as A, VALUE and B2 can be treated as single precision variables. There are times when a program uses mostly integers. In this case, it is inconvenient if % is added to each variable. The same goes for double precision and character variables. We have a very useful command for such occasions. With the DEF statement we can establish the variable type and leave out the sign.

```
DEF INT  variable name, variable name, ....
      SNG
      DBL
      STR
```

Let's type - DEF DBL A

The variable A is now regarded as a double precision variable. You do not have to type A# anymore. If you type -

```
DEF STR B
```

B is a character variable. For example, even if you type B= "Level3", the 'Type Mismatch' error message will not be

displayed.

If you set the form of the variable with DEF, for example - A, all the variables starting with A become the same variable type.

DEF STR A

If you have AB, AX, A(1)...., they all become character type variables.

It is a nuisance to set all the variable types, for example, writing all 26 characters in the alphabet so -

DEF DBL A, B, C,, Z

It is simply written if you designate the range A - Z.

With DEF DBL A - G

the variables A - G are all double precision.

* User Function Definition

The Level-3 has 49 functions. By combining these functions effectively we can perform mathematical calculation and handle characters easily.

When you use the same formula several times in the program, you must feel it is inconvenient if you had to re-type the same formula each time. Level-3 has a function to solve this problem.

DEF FN function name (argument string) = function
definition



Maximum 15 characters
The head of the character must
be alpha numerical

DEF FN is the abbreviation of DEFine FuNction. Look at the next example -

```
DEF FNX(X) = 70*SIN(X*3.14159/180)
```

When you use this in a program we make it -

```
A = FNX(1)
```

If the argument in () is the same as the X in the DEF statement, it does not matter whether it is a constant, a variable or a formula. The argument string is set temporarily to write the definition of the function. We do not have to use the set argument string.

```
5 WIDTH 40:SCREEN1
10 DEF FNX(X)=70*SIN(X*3.14159/180)
20 DEF FNY(Y)=30*COS(Y*3.14159/180)
30 FOR I=0 TO 360 STEP 4
40 PSET (160+FNX(I),100+FNY(I),7)
50 NEXT
```

If you execute this program a circle is drawn on the screen. Line numbers 10 and 20 define the function for calculating the position of the X and Y axes. The argument string X contains the degree parameters and the angle is converted into radians.

The loop in line numbers 30-50 gives the data from 0° - 360° in 4° lots. Line 40 PSET is the command to draw the circle. Here we use the function defined in lines 10 and 20.

Once you set a complicated function like this, it is useful to be able to use it again later. This method does not only apply to numerical functions but also to character functions and special functions.

Following are examples of a numeric argument string in a character function and a character argument string in a numeric function -

```
DEF FNA$(X)=B$ + CHR$(X)
```

↓

```
L$ = FNA$(I)
```

```
DEF FNA(X$) = (ASC(X$)-32)*32+&H2400
```

↓

```
K=FNA("M")
```

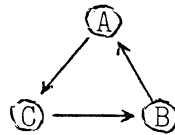
* SWAP - To exchange the values of two variables

Normally, when you exchange the values of two variables, for example A and B, three steps are necessary. You must first substitute A for a third value, C, then put B for A and C for B.

```
C = A
```

```
A = B
```

```
B = C
```



With the Level-3 SWAP command, two variable values can be exchanged in one go.

SWAP variable, variable

For example, we exchange A and B like this -

```
SWAP A, B
```

Note that the variables to be swapped must have the same form.

```
SWAP X$, Y$
```

```
SWAP A(I), A(J)
```

The following program is to input random numbers in some array variables, and change the order so that they are displayed from the smallest number up.

```

10 DIM A(10)
20 FOR I=0 TO 10
30 A(I)=RND(1)*100
40 NEXT
50 FOR I=0 TO 10
60 FOR J=I TO 10
70 IF A(I) > A(J) THEN SWAP A(I),A(J)
80 NEXT
90 NEXT
100 FOR I=0 TO 10
110 PRINT A(I)
120 NEXT

```

4. Error Procedures

The Level-3 has 38 error messages. When an error occurs, an error message together with the line number in which the error occurred is output and execution discontinues. Here we will explain how to handle errors.

* ON ERROR GOTO, RESUME - Error-handling routine

There are occasions when you do not wish to stop execution even if an error occurs (apart from errors caused by program mistakes).

```

10 INPUT A
20 INPUT B
30 C=A/B
40 PRINT C
50 GOTO 10

```

This is a division program. When $B = 0$, A cannot be divided. Therefore a 'Division by Zero' error would appear on the screen and the program execution would stop.

This error is not a program mistake but occurs during the execution of the program. In this case, we do not want to stop the program if we do not have to. So we can make use

of the ON ERROR GOTO statement -

ON ERROR GOTO line number

If you type this command as the first line of the program, control is transferred to the designated line number if an error occurs. We can handle the error easily this way.

```
10 ON ERROR GOTO 100
20 INPUT A
30 INPUT B
40 C=A/B
50 PRINT C
60 GOTO 20
100 PRINT"NOT DIVISIBLE"
110 RESUME 20
```

In this example, if B becomes 0, control is transferred to line number 100, 'Not Divisible' is printed without an error message occurring and control comes back to the input waiting condition.

RESUME is the command to return from the error-handling program to the original program. If you simply type -

RESUME

then control returns to the line in which the error originally occurred. If you do not remove the cause of the error at this time the control will execute the same error program again and again. With -

RESUME line number

control returns to the designated line number. In the above program, the control returns to line 20 and awaits input for variable A.

If you type RESUME NEXT,

control returns to the line after the one in which the error occurred.

* ERR, ERL - Error information

When the error occurs during the execution of the program, at times we do not know which line number and what error has occurred even if the control is transferred to the error-handling program. So we are unable to take the necessary steps. For this purpose, there are two functions - ERR and ERL by which we can find out the details of the error.

Let's look at this next program -

```
10 ON ERROR GOTO 100
20 A
30 END
100 PRINT "ERR CODE:";ERR
110 PRINT "ERR LINE:";ERL
120 RESUME NEXT
```

In line 20, as A is written alone it is obviously an error. The 'Syntax Error' message is displayed and its error code is 2.

In this program, when the error occurs, control jumps to line 100 with the ON ERROR GOTO statement and the error code and the line number containing the error are displayed. Using ERR and ERL, we can find out the error line number and its cause, and so we are able to take the necessary counter-measures.

* ERROR - Error generation

Even if we make an error-handling routine to fix any errors made, it will not, of course, be executed if there is no error. So we still do not know if that procedure works correctly or not. In Level-3, there is an ERROR command. This command simulates errors and is used to

check whether the error-handling procedure is working correctly. The BASIC outputs the error message corresponding to the error code.

ERROR n (n is the error code)

Study this next example -

```
10 ERROR 1
```

```
RUN
```

Next Without For In 10

An error defined by the error code as 1 has occurred.

This error command is inserted in the program first and executed. Then, if you confirm that the error routine is correctly executed, it is deleted. If you forget it and leave it in the program, an actual error will appear on the screen.

CHAPTER SIXMASTERING THE FUNCTIONS

In Level-3 we have three functions - a numerical function, a character function and a special function. Let's use these functions freely as if they were extensions of our hands.

1. Numerical Functions

It would be extremely useful if we could turn to one symbol when calculating trigonometrical functions and square roots, exponential functions and logarithms. The Level-3 has a number of functions for such numerical calculations.

* SIN, COS, TAN, ATN - Trigonometrical functions

There are four trigonometrical functions - SIN, COS, TAN and ATN (arc tangent). As these are all functions they are all described in the substituting statement and may not be used as independent commands.

```
A=SIN(X)      B=COS(X)
C=TAN(X)      D=ATN(X)
```

The argument unit which gives the value of the trigonometrical function is called the radian. When we express the values as degrees, we must convert from degrees to radians -

```
10 INPUT "Deg:",D
20 X=D*3.14159/180
30 Y=SIN(X)
40 PRINT Y
50 GOTO 10
```

The radian conversion is shown in line 20 and is found by degree $\times \pi / 180$. The output value is single precision with an effective column digit number of six columns.

Let's make the graphs of SIN, COS and TAN -

```

5 SCREEN 1
10 LINE(0,0)-(0,200),PSET,7
20 LINE(0,100)-(639,100),PSET,7
30 FOR I=0 TO 360
40 X=-70*SIN(I*3.14159/180)
50 X=100+X
60 PSET(I*1.5,X,7)
70 Y=-70*COS(I*3.14159/180)
80 Y=100+Y
90 PSET(I*1.5,Y,7)
100 Z=-70*TAN(I*3.14159/180)
110 Z=100+Z
120 IF Z<0 OR Z>200 THEN 140
130 PSET(I*1.5,Z,7)
140 NEXT

```

Line numbers 10 and 20 display the coordinate axes. The FOR-NEXT loop in line numbers 30 - 140 changes the angle from 0 degrees to 360 degrees and calculates the value of SIN, COS and TAN. It would be all right to show the results as they are but we multiply them by -70 to increase the amplitude and thus make the graph clearer.

Again, to obtain one cycle in the horizontal direction properly, we multiply by 1.5 in the PSET statement.

TAN is infinitive at 90° and 270° but since we cannot show this as a graph we set the limit in line 120.

Executing the program takes a little time, but the well-known graph appears on the screen. Trigonometrical functions are easily found just by giving the data to the argument.

There is another command - ATN. This is called the arc tangent and we use it to get the angle of a triangle. See Fig. 6.1.

Set Point B between the co-ordinates. Draw a perpendicular line from Point B down to the x axis. A is the point where this line meets the x

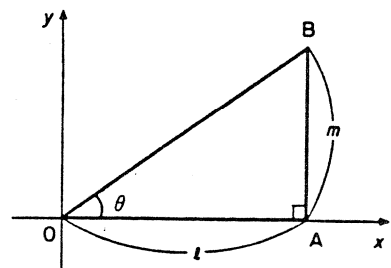


Fig. 6.1 Triangle

axis. We have now made the triangle OAB. We call the length of the base OA 'l' and the length of the perpendicular line BA m.

Let's find out if $\angle BOA = \theta$.

$$\tan \theta = \frac{m}{l}$$

$$\theta = \tan^{-1} \frac{m}{l}$$

$\tan^{-1}X$ is written in BASIC as ATN(X) and we input the numeral equivalent of the argument m/l into X. The result is obtained in radians. To convert to degrees, we multiply this value by $180/\pi$.

$$Y = \text{ATN}(X) * 180 / 3.14159$$

Let's make a program with this ATN command -

```

10 SCREEN 1:WIDTH 40
20 PRINT"** arctangent **"
30 LINE(200,50)-(200,100),PSET,7
40 LINE -(50,100),PSET,7
50 LINE -(200,50),PSET,7
60 LOCATE 13,5:PRINT"B"
70 LOCATE 13,13:PRINT"A"
80 LOCATE 1,13:PRINT"O"
90 LOCATE 13,9:PRINT"m"
100 LOCATE 7,13:PRINT"l"
110 LOCATE 34,8:PRINT"m"
120 LOCATE 18,9:PRINT" BOA = tan-1 ----"
160 LOCATE 34,10:PRINT"1"
170 LOCATE 0,16:INPUT"l:",L
180 INPUT"m:",M
190 X=ATN(M/L)*180/3.14159
200 LOCATE 25,12:PRINT"=";X;" "
210 LOCATE 0,20
220 END

```

* SQR - Square Root

To find the square root \sqrt{X} , we use the SQR(X) function. The usual form is -

A = SQR(X)

For example, when you want to find $\sqrt{2}$ -

PRINT SQR(2)

1.41421

Let's solve this second equation using the SQR(X) function -

$$ax^2 + bx + c = 0 \quad (a \neq 0, a, b, c \text{ are constants})$$

The solution is obtained with the following formula -

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The program for this is -

```

10 SCREEN 0
20 PRINT "** square root example **"
30 PRINT
40 PRINT "      ax^2+bx+c=0"
50 PRINT
60 INPUT "a:",A
70 INPUT "b:",B
80 INPUT "c:",C
90 F=B*B-4*A*C
100 IF F<0 THEN PRINT "overflow":END
110 H=SQR(F)
120 A2=A*2
130 ' x1(+)
140 X1=(-B+H)/A2
150 ' x2(-)
160 X2=(-B-H)/A2
170 PRINT
180 PRINT "x=";X1;"", ";X2

```

Line numbers 60 to 80 input a - c. First calculate the discriminant $D = b^2 - 4ac$. If it is negative, $SQR(X)$ becomes an error and this is judged by the IF statement in line 100.

The calculated values for $\sqrt{b^2 - 4ac}$ and $2a$ are entered into H and A2 respectively. We find the two values of X. In line 140, the positive $(-b + \sqrt{b^2 - 4ac}/2a)$ is calculated and in line 160, the negative $(-b - \sqrt{b^2 - 4ac}/2a)$ is calculated.

* EXP - Exponential function

This gives the EXPonent of the argument.

$$\text{EXP}(X) = e^x$$

PRINT EXP(5)

148.413

The overflow message appears if you input an argument of more than 89.

* LOG - Natural logarithms

This function calculates the value of the natural logarithm of X.

$$\text{LOG}(X) = \log eX$$

Let's calculate $\log e 5$ -

PRINT LOG(5)

1.60944

LOG(5) is 1.60944 (that is, $e^{1.60944} = 5$). You can confirm this -

PRINT EXP(1.60944)

5.00001

We would expect this to be 5 but due to a calculation error in conversion by significant column number (of 7), it is 0.00001 over.

To avoid mis-conversion errors place LOG in EXP() -

```
PRINT EXP(LOG(5))
```

```
5
```

* Common logarithms

As Level-3 does not have a function for common logarithms the calculation is based on natural logarithms. The base of a common log is 10 so the relationship of the two logarithms is written as -

$$\log_{10} X = 0.434295 \log_e X$$

For example, when $\log_{10} 3$ is the value of X at $10^X = 3$, we can calculate as follows -

```
PRINT LOG(3)*0.434295
```

```
.477122
```

Let's reverse this and check it -

```
PRINT 10^.477122
```

```
3.00001
```

The result contains a conversion error but it is practically the same as the original 3.

* ABS, SGN - Absolute value

We use ABS(X) to obtain the absolute value of a number.

```
A = ABS(X)
```

$ABS(X) = |X|$ means ABS gives the absolute value of the X with the sign removed.

```
PRINT ABS(-10)
10
```

When you want to check the numeral's sign, we use the $\text{SGN}(X)$ function. This function decided whether the X is positive or negative.

```
A = SGN(X)
```

The value of the $\text{SGN}(X)$ function becomes -1 if X is negative, 0 if X is 0 and 1 if X is positive.

```
PRINT SGN(3);SGN(0);SGN(-30)
1 0 -1
```

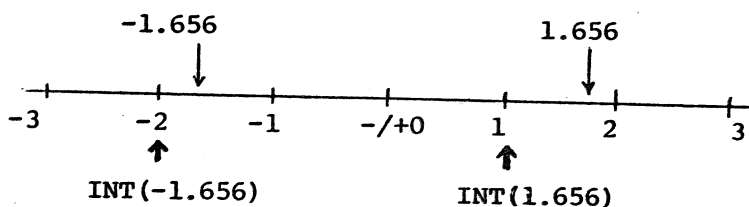
* INT, FIX, CINT - To make integers

Level-3 has three functions for making integers.

```
(1) A = INT(X)
```

$\text{INT}(X)$ (INTEger) gives the maximum integer not exceeding X . For example, 1.656 becomes 1 and -1.656 becomes -2.

```
PRINT INT(-1.656)
-2
```



```
(2) A = FIX(X)
```

The $\text{FIX}(X)$ function disregards the decimal part of a number. In other words it gives only the integer part of X .

```
PRINT FIX(1.656),FIX(-1.656)
1 -1
```


In the preceding example, -1.656 becomes -1.

(3) $A\% = \text{CINT}(X)$

CINT (Convert INTegeR) converts X to an integer. This conversion rounds the first digit after the decimal point.

$\text{PRINT CINT}(1.4999), \text{CINT}(1.5111)$

1 2

* CSNG, CDBL - Conversion function

There are three functions for converting integers as we have already mentioned. There are also other functions to convert single precision and double precision to numerical values. You may feel this has no meaning but we can minimise errors during conversion by using these functions.

(1) $A = \text{CSNG}(X\#)$

CSNG (Convert SiNGle) converts integers or double precision numerical value X to single precision.

(2) $A = \text{CDBL}(X)$

CDBL (Convert DouBLe) converts integers and single precision X to double precision.

$A\# = \text{CDBL}(I)$

$A = \text{CSNG}(J\#)$

You can use $A = I\#$ (single precision = double precision) or $A\# = I$ (double precision = single precision) to convert and no error will occur. However, you will not be able to estimate conversion errors. We therefore recommend that you use CSNG and CDBL functions.

* RND - Random numbers

```
A=RND(1)
```

A table made from numbers which have an equal probability of occurring is called a random table. In Level-3, the RND(X) function generates a range of random numbers between 0 and 0.999999.

```
10 PRINT RND(1)
```

```
20 GOTO 10
```

If you RUN this program several numbers between 0 - 0.999999 are output. Try hitting **BREAK** to stop, then starting again with RUN. If you watch carefully, you will see that the same random number occurs. By designating RND(1), Level-3 automatically returns the random number from the first page of its table.

If you execute RND(X) when argument x is -n, the Level-3 moves to page n of its random number table.

When we then execute RND(1), random numbers from page n are generated. When we execute RND(0), one value is selected from the fixed random numbers on page n. Accordingly, RND(0) has the same value.

RND (1) or RND ← outputs random numbers in order
from that page

RND (-n) ← transfers the random number
table to page n

RND (0) ← takes the number at the head of
this page as the value

Random numbers are often used in games but numerals with decimal places are not much use. Let's make them a little easier to use.

Instead of throwing dice, our next program generates integers between 1 and 6 using RND.

```
10 D=RND(1)*6
20 DIE=INT(D)+1
30 PRINT DIE
40 GOTO 10
```

Since the random numbers are between 0 - 0.999999, when we multiply by 6, we arrive at new numbers between 0 - 5.999999. One of them is substituted for D (line 10).

We change this to an integer with INT and obtain an integer between 0 - 5. We add 1 to this to make it the same as a die and substitute it for the variable DIE (line 20). We then PRINT it on the screen (line 30).

In this way, if we multiply RND(X) by n (maximum value of the integer you want to generate), we can convert to integers and end up with an integer between 0 - n - 1.

You can also convert to integers using the FIX function instead of the INT function.

2. Character Functions

Character functions are functions for handling characters and character strings and are different to the functions which you use in mathematics.

* LEFT\$, RIGHT\$, MID\$ - Handling character strings

Let's assume the character string ABCDEFG is in A\$. To transfer the contents of A\$ to B\$ we use B\$ = A\$.

But you may only want to substitute a part of the contents in A\$. In this case, we use the above three statements. For example, to input the 3 characters on the left of ABCDEFG into B\$, we use the LEFT\$ function and designate -

```
B$ = LEFT$ (A$, 3)
```

With this statement, the three characters on the left in A\$ (that is, ABC) are transferred to B\$.

How do we input the four right-hand characters in A\$? We use the RIGHT\$ statement -

```
B$ = RIGHT$ (A$, 4)
```

and the characters DEFG are substituted for B\$.

When you want to input the middle characters BCDE, we use the MID\$ statement -

```
B$ = MID$ (A$, 2,4)
```

This statement extracts four characters starting with the second character from inside variable A\$.

To summarise -

```
Character variable = LEFT$(character variable, length)
Character variable = RIGHT$(character variable, length)
Character variable = MID$(character variable, nth
                        character from the left, length)
```

We have shown an example of each of the functions below. Note their differences.

```
10 A$="BASIC MASTER LEVEL3"
20 FOR I=1 TO 19
30 PRINT LEFT$(A$,I)
40 NEXT
```

```
10 A$="BASIC MASTER LEVEL3"
20 FOR I=1 TO 19
30 PRINT RIGHT$(A$,I)
40 NEXT
```

```
10 A$="BASIC MASTER LEVEL3"  
20 FOR I=1 TO 19  
30 PRINT MID$(A$,I,1)  
40 NEXT
```

* LEN - To check the length of the character string

```
A=LEN(X$)
```

The LEN(X\$) function gives the number of characters making up the character string (with no relation to the contents of X\$).

```
10 A$="BASIC"  
20 L=LEN(A$)  
30 PRINT L
```

LEN has a numerical value so we must substitute a numerical variable. If A\$ has no value (that is, A\$=" ") the value of LEN(A\$) becomes 0.

* ASC, CHR\$ - Code conversion

As mentioned before, ASCII code has been allocated to all the characters used in BASIC. We use the ASC function when we want to see the character code. ASC gives the converted value of the ASCII code of the first character in the character string. Even if many character strings are given to the ASC function argument, only the first character at the head is converted to character code.

The reverse function, that is, to generate characters from the character code is performed by CHR\$(X). The character value which corresponds to character code X is given by this function.

```
A$=CHR$(X)
```

The PRINT statement only accepts character input from the keyboard but a new code, that is a code using the CTRL key, may also be generated by using CHR\$.

```
10 FOR I=0 TO 255
20 PRINT CHR$(I);
30 NEXT
```

* STR\$, VAL - Numeric character conversion

Converting numerical values to character numerals or vice versa is programmed by using -

```
A$=STR$(X)
A=VAL(X$)
```

STR\$ stands for string and it converts the numerical value of the argument as it is to a character -

```
10 A=128
20 S$=STR$(A)
30 PRINT S$
```

By handling numerical values as characters, we can carry out various processes with other character functions and thus the range of applications is broadened.

The reverse of STR\$ is VAL(X\$). If the contents of X\$ is a character number, it will be transformed as is to a numerical value. If the X\$ value is a character other than a numeral, it becomes 0. Further, if a character other than a numeral appears in the middle of the numerals, the numerals so far are converted to numerical values.

```
10 A$="12345"
20 B=VAL(A$)
30 PRINT B
```

* SPACE\$, STRING\$ - Character generator function

To insert spaces in a character variable we use -

A\$ = " _ _ _ "

Note the spaces are enclosed by ("). However, if there are a lot of spaces we can use the function SPACE\$(X) especially for this. This function generates only the number of spaces designated in the argument -

A\$=SPACE\$ (number)

Also, to generate a certain character repeatedly as well as spaces, we have the STRING\$(X,Y) function. For example, when you input a hundred A characters in A\$ -

A\$=STRING\$ (100, "A")
 ↑
 designated character

As the designated character, we can use a character variable, a numerical constant, a numerical variable or a formula as well as a character variable.

* INSTR - Searching function

When you make data base programs, you often need to know if a certain character is in particular character string. The INSTR function is used for this purpose. This is the general format -

A=INSTR (I, X\$, Y\$)

X\$ is the original character string and Y\$ is the character string to be searched. I may be omitted. If designated, the search starts from the first character in X\$. If Y\$ is not included in X\$, the function value is 0. If it is, the function value is that of whichever character it finds.

```

10 A$="BASIC MASTER"
20 D=INSTR(A$,"MAS")
30 PRINT D

```

* INKEY\$, INPUT\$ - Input function

We normally use the INPUT statement for keyboard input but if we want to see the condition of a particular key, we use INKEY\$. This is useful for games. This INKEY\$ does not have an argument.

```
I$=INKEY$
```

Pressing a key during the execution of INKEY\$ gives that character as the function value. If no key is pressed we get a null character and the program continues. This function does not wait for input.

```

10 A$=INKEY$
20 IF A$="" THEN 10
30 PRINT A$

```

INPUT\$ is used when you take n characters from the buffer after opening the file. For example -

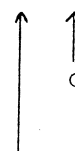
```

10 OPEN"I",#1,"KYBD:"
20 K$=INPUT$(2,#1)
30 PRINT K$
40 CLOSE

```

The usual format is -

```
INPUT$(X, Y)
```



 ↑

 opened file number

 ↑

 x characters

To execute this, we must open the file first. In this program, the keyboard is opened in line 10. The INPUT\$ designates taking 2 characters from file No. 1. These two characters are input to K\$ and displayed on the screen with PRINT for confirmation. Then we close the file.

If you press RUN, the cursor flickers. When you press any two keys, those characters appear on the screen.

* HEX\$, OCT\$ - Hexadecimal and octal conversion

When you make a simple machine language monitor program in BASIC, you may want to display numbers as hexadecimal or octals.

Hexadecimal conversion is HX\$=HEX\$(X)

Octal conversion is OC\$=OCT\$(X)

This converts the argument into characters. X is an integer between 0 - 65535. When you change hexadecimal or octals back to the original number, use VAL.

```
PRINT HEX$(65535)
```

```
FFFF
```

```
PRINT OCT$(65535)
```

```
177777
```

As hexadecimal and octal - &Hnnnnn or &Onnnnnn

Thus - A=VAL("&H"+H\$)

↑

character showing hexadecimal

or

A=VAL("&O"+O\$)

↑

character showing octal

and we can change to a numerical value.

3. Special Functions

* POS, CSRLIN - Cursor functions

To find out the position of the cursor on the screen, we have a cursor position detect function.

For horizontal position detection, the POS(0) function is used and for vertical position detection, CSRLIN is used.

(1) POS

X=POS (0)

POS stands for POSition. When the POS argument is 0, the cursor's horizontal position on the screen is given.

```
10 LOCATE 20,10
20 X=POS(0)
30 PRINT X
```

When you want to see the printer head position, designate the POS argument for the file number you have opened.

```
10 OPEN"0", #1,"LPT0:"
20 PRINT#1,"ABCDEFGHI";
30 X=POS(1)
40 PRINT X
50 CLOSE
```

By setting the file descriptor at SCRn: or COM0: we are able to see the cursor position at the output file. If you do not have a printer, change the file descriptor in line 10 LPT0: to SCRn: and change line 30 to X=POS(0). Then try executing it.

(2) CSRLIN

Y=CSRLIN

CSRLIN stands for CurSoR Line and gives the cursor vertical position. This function does not have an argument so do not mistake it for a variable.

```
10 LOCATE 0,15
20 Y=CSRLIN
30 PRINT Y
```

* POINT, SCREEN - Screen control function

(1) POINT

This function is for checking the presence of a dot on certain co-ordinates when we are controlling graphics.

```
P=POINT(X, Y)
```

If there is a dot at the designated X, Y position, -1 is substituted for P. If there is no dot, 0 is substituted. The range for X is from 0 - 639 and the range for Y is from 0 - 199.

```
10 PSET (0,0)
20 P=POINT(0,0)
30 PRINT"PSET "; P
40 PRESET(0,0)
50 P=POINT(0,0)
60 PRINT"PRESET";P
```

(2) SCREEN

To take one character on the screen or to check whether a character or a graphic is at a designated position, we use the SCREEN function.

This function has the same spelling as the SCREEN statement (for setting the screen) so take care not to confuse them.

```
C=SCREEN(X, Y, 0)
    ↑   ↑   ↑
hor. pos. vertical position switch
```

To take one character code from the screen designated by X, Y, set the switch to 0. One character code is substituted for C. Use CHR\$ to convert it to a character again. If you set the switch to 1 then the attribute of the position is returned.

```
C=SCREEN(X,Y,1)
```

A numerical value is substituted for C. If after dividing it by 16, the quotient is an even number, it is defined as a character and if it is an odd number it is defined as a graphic. The remainder left after dividing it by 16 becomes the colour code.

```
10 SCREEN0
20 LOCATE 0,0
30 PRINT "A"
40 C=SCREEN(0,0,0)
50 PRINT"CODE:";C
60 PRINT"CHAR:";CHR$(C)
```

```
10 SCREEN0
20 LOCATE 0,0
30 COLOR 5
40 PRINT"A"
50 C=SCREEN(0,0,1)
60 PRINT"GR/CH:";FIX(C/16)
70 PRINT"COLOR:";C MOD 16
```

In this way, we can readily grasp the state of a part of the screen.

* SPC, TAB - Functions in a PRINT statement

There are two functions which can be used in a PRINT statement. The first one is SPC(X) -

```
PRINT SPC(X) ;
```

SPC outputs X spaces from the existing cursor position.

```
10 PRINT"A";SPC(10);"B"
```

This program writes A first, then outputs 10 spaces and writes B.

The second function is TAB(X). TAB moves the cursor to the Xth character position from the head character on that line and PRINTs.

```
PRINT TAB(X) ;
```

This function is often used to order the head of variable data when displayed. X can be set from 0 - 39 columns when in 40-ch. mode and 0 - 79 columns when in 80-ch. mode.

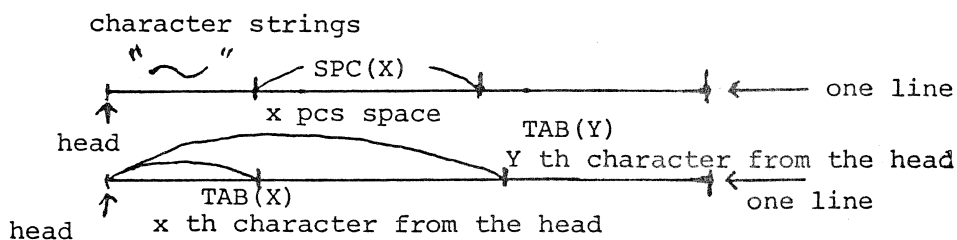


Fig. 6.2 SPC and TAB

```
10 WIDTH
20 SCREEN0
30 A=INT (RND(1)*1000)
40 B=A/16
50 C=FIX(B)
60 PRINT A;TAB(10);B;TAB(25);C
70 GOTO 30
```

* TIME\$, TIME, DATE\$, DATE - Clock, calendar functions

In Level-3, there are built-in clock and calendar functions. When you want to see the clock -

```
PRINT TIME$
```

TIME\$ is displayed as a 24-hour day. However the moment the power is put on becomes 0 hours 0 minutes 0 seconds. You must therefore set it to the present time otherwise only the total time from when the power was turned on is displayed.

```
PRINT TIME$
```

```
00:00:09
```

```

↑   ↑   ↑
|   |   |
|   |   | seconds
|   |   |
|   |   | minutes
|   |   |
|   |   | hours

```

To have this function as a clock, first of all we must set it to the present time as follows -

```
TIME$ = "11:59:00"
```

TIME\$ is a character-type so enclose the time with ("). The clock now functions separately from the program and the time output is the present time.

We also have a numerical-type TIME. This function measures the total time elapsed from 00:00:00 up to the present. We use TIME as a numeral and thus can perform calculations with it.

The date is displayed by DATE\$ or DATE.

```
PRINT DATE$
```

```
00/00/00
```

```

↑   ↑   ↑
|   |   |
|   |   | day
|   |   |
|   |   | month
|   |   |
|   |   | year

```

When TIME\$ exceeds 23:59:59, one day is added to the date. When the power is first turned on the date is 00 year 00 month 00 day. This means we must set the date like we did with TIME\$ if we wish to use this function as a calendar.

To delimit TIME\$ we use (:) but for DATE\$ we use (/).

```
10 DATE$="81/02/28"
20 TIME$="23:59:59"
30 FOR I=0 TO 1000:NEXT
40 PRINT DATE$.TIME$
```

The result of this program is 1 March 1981. 1980 was a leap year, so if we run -

```
10 DATE$="80/02/28"
20 TIME$="23:59:59"
30 FOR I=0 TO 1000:NEXT
40 PRINT DATE$.TIME$
```

We get 29 February 1980 so we can see that the leap year is also being controlled correctly.

Let's now make a digital clock by using these functions in the following way -

```
10 WIDTH 40
20 SCREEN 0
30 INPUT"DATE:(YY/MM/DD)",D$
40 DATE$=D$
50 LINEINPUT"TIME:(HH:MM:SS)",T$
60 TIME$=T$
70 CLS
80 LINE(120,50)-(400,100),PSET,4,B
90 LOCATE 10,B
100 PRINT "DATE: ";DATE$
110 LOCATE 10,10
120 PRINT "TIME: ";TIME$
130 GOTO 90
```

DATE\$ and TIME\$ cannot be used as input variables with the direct INPUT statement so we must substitute another variable for them. We cannot use (:) in the INPUT statement either so we use LINE INPUT for the TIME\$ function.

* PEN - Light pen function

The Level-3 is equipped with a light pen interface as a standard. More detailed information follows later but we will introduce the functions of the light pen here.

There are two kinds of function groups to control the light pen. One is called the trigger sense group and the other is the level sense group.

(1) Trigger sense

This gives information when the pen is pressed to the screen. To read the next information you must first remove the pen from the screen and then press it to the screen again.

PEN(0), PEN(1), PEN(2)

When the argument is 0, it gives the function indicating whether the pen is pressed or not. When the pen is pressed, it gives -1 and if not, it gives 0.

PEN(1) and PEN(2) are the value of the X and Y co-ordinates.

```
10 SCREEN 0
20 WIDTH 80
30 COLOR 1,7
40 IF PEN(0)=0 THEN 40
50 LOCATE PEN(1),PEN(2)
60 PRINT"X"
70 GOTO 40
```

The range for the horizontal position PEN(1) is 0-79 characters and the vertical position PEN(2)'s range is 0-24.

(2) Level sense

This enables us to read the X,Y co-ordinate information as long as the pen is held to the screen. Therefore it is

possible to read each different co-ordinate by sliding the pen over the screen.

The level sense is controlled by -

PEN(3), PEN(4), PEN(5)

PEN(3) has the information on whether the pen is being pressed or not. It gives -1 when being pressed and 0 when not being pressed.

PEN(4) gives the horizontal position and PEN(5) the vertical position.

When you move trigger sense to level sense change the program as follows -

PEN(0) → PEN(3)

PEN(1) → PEN(4)

PEN(2) → PEN(5)

* ERR, ERL - Error-handling routine

When an error occurs during program execution, these two functions advise us where and what error has occurred. The detection of the error is carried out by the ON ERROR GOTO statement. For more details refer back to pp. 89.

* FRE - To check the usable memory area

When you are making a program you may want to know how much FREe memory is left. The FRE(X) function will supply you with this information.

PRINT FRE(0)

14172

The number displayed by the PRINT statement is the remaining number of usable bytes. The argument 0 is just a dummy. Any number would be acceptable.

When you want to see the remaining character area, designate a character in the argument. Character variables may also be written.

```
PRINT FRE("A")
```

```
300
```

* PEEK - To check the data in the memory

This function is used to look at one byte in the memory.

```
A=PEEK(X)
```

The PEEK argument X is the address. The data from one byte in the memory designated by this address is stored in A. For the variable any decimal from 0-255 is acceptable.

Following is an example of a memory dump program using PEEK -

```
10 SCREEN 0
20 WITH 80
30 INPUT "Sta add (HEX):",A$
40 INPUT "End add (HEX):",B$
50 B=VAL("&H"+B$)
60 A=VAL("&H"+A$)
70 H$=RIGHT$("0000"+HEX$(A),4)
80 PRINT H$;" ";
90 FOR I=0 TO 15
100 D=PEEK(I+A)
110 D$=HEX$(D)
120 PRINT RIGHT$("00"+D$,2);" ";
130 IF A+I >= 3 THEN 180
140 NEXT
150 PRINT
160 A=A+1
170 GOTO 70
180 PRINT
190 GOTO 30
```

The start and end address are input by hexadecimal, and we converted to numerals in line numbers 50-60. Line 70 converts the address to a 4-digit hexadecimal and outputs it in line 80. To display 16 bytes in one line, FOR is defined in line 90. In line number 100, data is taken from the memory and converted to a 2-digit hexadecimal in line numbers 110-120.

In this way it is possible to examine the memory quite easily. This PEEK function is for reading the contents of the memory and the POKE statement has the reverse function.

CHAPTER SEVENGRAPHICS1. BASIC GRAPHICS* Normal Mode & High Resolution Mode

One of the special features of Level-3 is its superior colour graphic function. There are two graphic modes - one is normal and the other is high resolution mode.

High resolution mode is equivalent to bit image which makes it possible to draw complicated graphics. Sometimes normal mode may be called low resolution mode in comparison with high resolution mode.

The state of the Level-3 screen has four stages depending on the size of the dot -

80 x 100, 160 x 100 - Normal mode
320 x 200, 640 x 200 - High resolution mode

These four graphic modes are designated by a combination of the WIDTH and SCREEN functions.

* WIDTH, SCREEN - Set graphic mode

WIDTH sets the number of horizontal display characters and selects either 40- or 80-character mode. A 40-ch. mode character is double the size of that of the 80-ch. mode.

WIDTH 80 → 80-character mode

WIDTH 40 → 40-character mode

SCREEN has the following three functions -

- switches normal and high resolution mode
- designates the page if multipage
- switches interlace and non-interlace

```

SCREEN  l,  m,  n
        ↑    ↑    ↑
        |    |    |
        |    |    | interlace changeover
        |    |    | 1 or 0
        |    |    |
        |    |    | multipage
        |    |    | designation
        |    |    |
        |    |    | graphic
        |    |    | mode
        |    |    | designation
        |    |    | 1 or 0

```

'l' designates the graphic mode. If it is 0, it designates normal mode and if it is 1, high resolution mode.

'm' designates the page number. 'n' is to switch interlace. 0 is in interlace mode and 1 is in non-interlace mode.

Each parameter can be omitted depending on the purpose. If it is, the previous state is continued.

- (1) To change the graphic mode only (clears the screen) -
SCREEN 1 or SCREEN 0
- (2) To change only the page -
SCREEN, 1 or SCREEN, n
- (3) To change interlace mode (screen not cleared) -
SCREEN,, 0 or SCREEN,, 1

Note that if the graphic mode is set with a SCREEN command, the screen is cleared.

Often the 'Illegal Function Call' error message appears on the screen when we use WIDTH and SCREEN commands. This happens when the mode we try to set with WIDTH and SCREEN does not correspond to the screen mode when the power was switched on.

For example, set the mode switch to 0 and turn the power on. In this mode, a maximum screen memory of 8 Kbyte is set. This is the same as when we set WIDTH 40, SCREEN 1. What would happen if we tried to execute WIDTH 80?

This is the designation for 640 x 200 graphic mode but the necessary screen memory is 16 Kbytes.

Requesting 16 Kbytes when only 8Kbytes has been set is physically impossible. And so, an error will occur. We must understand the necessary screen memory size when we reset the mode with the internal dip switch or with a NEW ON statement.

Table 7.1 shows the memory capacity of one screen. If the power is turned on at MODE 0, an 8 Kbyte memory can be used on the screen. If the power is put on at MODE 1, a 16 Kbyte memory can be used. We almost always select one of these, so we must consider this relationship when making the program.

	SCREEN 0		SCREEN 1	
	graphics x × y	memory	graphics x × y	memory
WIDTH 40	80 × 100	1K	320 × 200	8 K
WIDTH 80	160 × 100	2K	640 × 200	16K

↑
↑
 normal resolution high resolution

Table 7.1 WIDTH and SCREEN Relationship

SCREEN 0 is the normal setting but how would we go about setting graphics with this mode?

320 x 200, 640 x 200 have complete bit image. 128,000 dots are required to light all the dots in 640 x 200 mode. However, in normal mode, we have a method called character division whereby graphics can be displayed with only a small memory.

Normally a character is made up of 8 x 8 dots in non-interlace mode (8 x 16 dots in interlace mode). But when displayed as a graphic, we make a character divided by two in the horizontal and divided by four in the vertical direction and eight dots are lit.

VRAM (video RAM) requires one byte to display one character but, to display graphics, one byte is needed also to light up these eight dots.

At this time, each bit corresponds with each lighting dot. The combination of the 8 dots has a total of 256 varieties and is the same as the combination which can be displayed by one byte. It is not possible to divide the character into smaller portions.

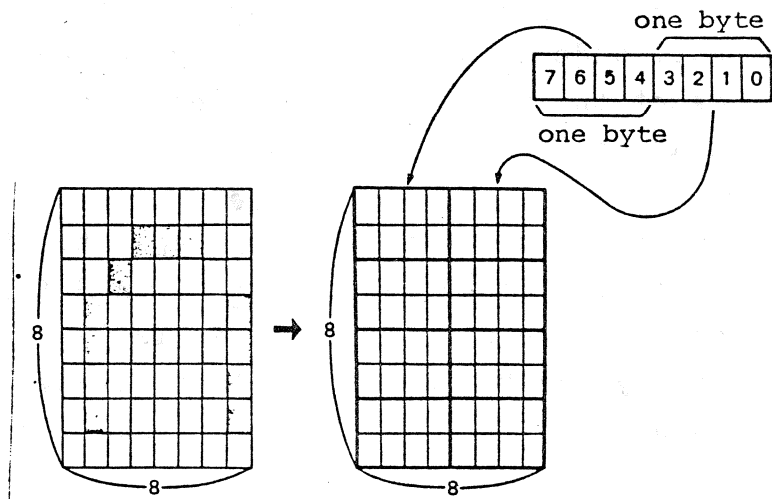


Fig. 7.2 Normal Mode Graphic Character

* Colour Code

Eight colour graphics are possible in the Level-3. The colour designation is carried out by the corresponding number of each colour. Let's recall the colour and its corresponding number -

0 : black	4 : green
1 : blue	5 : cyan (light blue)
2 : red	6 : yellow
3 : magenta	7 : white

We give a concrete example of colour designation below.

* PSET, PRESET - Placing the point and erasing it

We use the PSET command to display one point in a designated position. PSET stands for Point SET.

PSET (X, Y, C)

The range of the horizontal position X is 0-639 and the vertical position is 0-199. C is the colour code to designate points of several colours. Let's make the program.

Set the MODE switch to 1 and turn the power on.

```

10 CLS
20 X=INT(RND(1)*320)
30 Y=INT(RND(1)*100)
40 C=INT(RND(1)*8)
50 PSET(320-X,100-Y,C)
60 PSET(320-X,100+Y,C)
70 PSET(320+X,100+Y,C)
80 PSET(320+X,100-Y,C)
90 GOTO 20

```

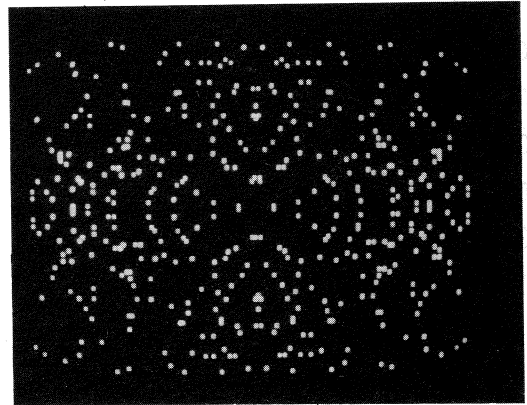


Photo 7.3 PSET Execution Example

When the program is executed, as shown in Photo 7.3, the points are plotted from the centre to right and left, up and down, symmetrically.

As you can understand from this program, you can use formulae, variables, constants, etc. freely.

The original background colour is black so that if black is used in PSET, it seems as if nothing has happened. However, as designated, black points have actually been drawn. The same thing applies when you change another colour to black. It looks as if that point has disappeared but actually the black point exists.

Let's talk about PRESET (Point RESET) next. This command erases the points.

PRESET (X, Y)

The respective ranges of X and Y are 0-639 and 0-199 the same as PSET. In the previous program, we used `C=INT(RND(1)*8)` in line 40 to designate the colour. As C is an integer from

0-7 this program also produces black which appears to erase the other colours.

Let's change line number 40 to $C = \text{INT}(\text{RND}(1) * 7) + 1$ so as not to generate 0.

```

10 CLS
20 X=INT(RND(1)*320)
30 Y=INT(RND(1)*100)
40 C=INT(RND(1)*7)+1
50 PSET(320-X,100-Y,C)
60 PSET(320-X,100+Y,C)
70 PSET(320+X,100+Y,C)
80 PSET(320+X,100-Y,C)
90 X=INT(RND(1)*320)
100 Y=INT(RND(1)*100)
110 PRESET(320-X,100-Y,C)
120 PRESER(320-X,100+Y,C)
130 PRESET(320+X,100+Y,X)
140 PRESET(320+X,100-Y,C)
150 GOTO 20

```

Line numbers 90-150 make up the added part. This program produces four dots and erases four dots at the same time. This program and the previous one do not appear to be different. However the colouring is a little different.

Level-3's maximum resolution is 640 x 200 dots but the colour resolution is not as accurate as this. Also the horizontal line is in character units. The relationship between dot and colour is shown in Table 7.4.

Table 7.4 Level-3 Colour Resolution

Dot Res.	Colour Res.
640 x 200	80 x 100
320 x 200	40 x 100
160 x 100	80 x 25
80 x 100	40 x 25

High resolution

Low resolution

High resolution mode has bit image. Vertically its resolution is as it stands but horizontally, it has a colour resolution of 80 at 640 dots and 40 at 320 dots. That is to say, every eight dots have the same colour.

Up against this, normal mode low resolution makes use of the character resolution method and so the colour resolution is in character units both vertically and horizontally.

We will outline the difference between placing a black dot on the screen and erasing dots using PRESET.

Let's imagine that we have cut one byte off the screen. We will assume that there are three red points in one byte. If we erase the red point 7 we designate black to this point. Since the colour appears in character units the whole byte becomes black and we cannot see it anymore.

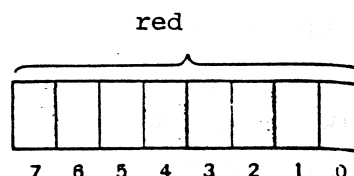


Fig. 7.5 1 Byte Colour Display

However this is different to erasing dots with PRESET. The three points, 7, 4 and 2 are still there as before. This is confirmed by the fact that if you make the 0 position a blue dot, points 7, 4 and 2 also become blue.

* COLOR - Colour designation

The COLOR statement sets colour.

```

COLOR m, n
  ↑   ↑
  |   |
  |   | background colour
  |   | character colour

```

m is the character colour and n the background colour. When you want to change only the character colour we write for example, -

```
COLOR 4
```

With this, the character becomes green. If you want to change only the background to white, the following command is used -

COLOR, 7

To change both we input -

COLOR 4, 7

Try to execute the above program setting a colour other than black with the COLOR command.

* LINE - Drawing a line

Let's try drawing a line next.

To draw a line using PSET is somewhat difficult. Level-3 has a LINE command which draws a line between a designated starting point and finishing point. We use this -

LINE(X ₁ , Y ₁)-(X ₂ , Y ₂),	PSET	C,	B
	<u>PRESET</u>		BF
↑	↑	↑	↑
starting	ending point	colour	option
point			
	function		

In the LINE statement, we designate PSET or PRESET as well as the starting and finishing point. It is the same as drawing points. We draw the line with PSET and erase it with PRESET. C is the colour designation and decides the colour of the line. We will explain the B and BF later. For the moment, however, look at the example -

LINE(0, 0) - (639, 199), PSET, 4

When we execute it directly, a green line from the upper left corner to the bottom right is drawn. The LINE statement draws a straight line from any position by designating two points on the screen.

Of course, it is possible to designate formulae, variables and constants for X, Y and C. The program below combines all these -

```

10 CLS
20 C=INT(RND(1)*7)+1
30 GOSUB 100:GOSUB 200
40 GOTO 20
100 FOR Y=0 TO 199 STEP 4
110 LINE(0,Y)-(Y*3,199),PSET,C
120 NEXT
130 FOR Y=199 TO 0 STEP -4
140 LINE (Y*3,0)-(640,Y),PSET,C
150 NEXT
160 RETURN
200 FOR Y=0 TO 199 STEP 4
210 LINE(0,Y)-(Y*3,199),PRESET
220 NEXT
230 FOR Y= 199 TO 0 STEP -4
240 LINE(Y*3,0)-(640,Y),PRESET,C
250 NEXT
260 RETURN

```

Lines 100-160 form a PSET subroutine. Lines 200-260 form the PRESET subroutine for erasing. Hit RUN then you should get the pattern shown in Photo 7.6.

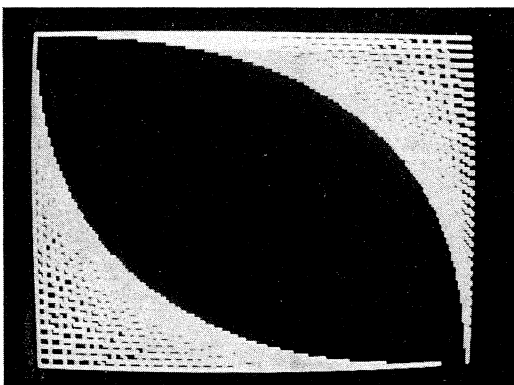


Photo 7.6 Example 1 of Program using LINE statement

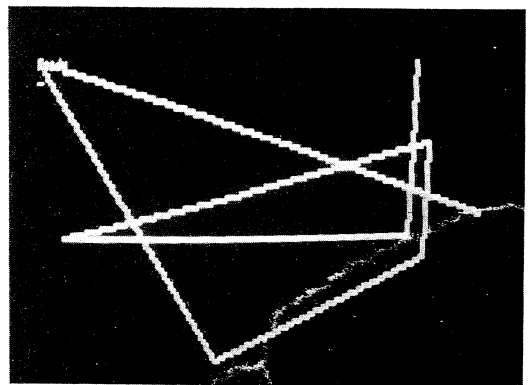


Photo 7.7 Example 2 of Program using LINE

The next program combines PSET, PRESET and LINE statements. Lines 10 and 20 designate the start. Line numbers 30-90 set an infinite loop. Photo 7.7 shows the result of the program.

```

5 CLS
10 X=INT(RND(1)*640):Y=INT(RND(1)*200)
20 X1=INT(RND(1)*640):Y=INT(RND(1)*200)
30 C=INT(RND(1)*7+1)
40 LINE(X1,Y1)-(X,Y),PSET,C
50 X2=INT(RND(1)*640):Y2=INT(RND(1)*200)
60 LINE(X,Y)-(X2,Y2),PSET,C
70 LINE(X1,Y1)-(X,Y),PRESET
80 X1=X:Y1=Y:X=X2:Y=Y2
90 GOTO 30

```

Now, look at the next program. There are no starting point parameters in the LINE statement. You would think an error would occur if you executed this program but it proceeds normally. In fact, this is possible because the starting point value has been input as previously executed end point value designated by the LINE statement. This function is useful for making a continuous line chart.

```

10 CLS
20 C=INT(RND(1)*7+1)
30 LINE -(RND(1)*640,RND(1)*200),PSET,C
40 GOTO 20

```

Let's make a program for a continuous line chart using this function -

```

10 CLS
20 LINE (0,0)-(0,199),PSET,7
30 LINE -(639,199),PSET,7
40 J=10:C=4:GOSUB 100
50 J=50:C=2:GOSUB 100
60 J=100:C=1:GOSUB 100
70 END

```

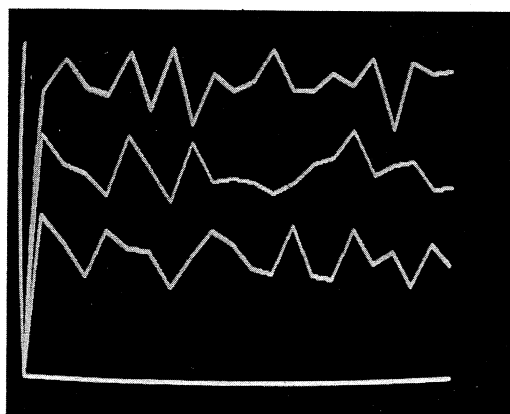


Photo 7.8 Drawing a Continuous Line Chart

```

100 LINE(0,199)-(0,199),PSET,C
110 FOR I=29 TO 639 STEP 30
120 LINE -(I,RND(1)*50+J),PSET,C
130 NEXT
140 RETURN

```

Lines 20-30 display the co-ordinates. Numbers 100-140 make up the subroutine to actually draw the graph. Data provided from the main program is : the colour code set in C, and the numerical value to change the direction of the vertical axes which is set in J. The LINE statement in line 100 provides the starting point for the next LINE statement after setting the origin of the co-ordinate axes (0, 199). The FOR statement in line 110 decides the width of the graph.

Thus if you make the STEP 30 smaller, the number of turning points increases. The height of the graph is given in random numbers. If you want to plot meaningful numbers, you must change the LINE statement in line number 120.

We can also make function graphs -

```

10 CLS
20 LINE (0,100)-(639,100),PSET,7
30 LINE(316,0)-(316,199),PSET,7
40 FOR I=0 TO 639 STEP 3
50 X=-70*SIN(I/100)+100
60 PSET(I,X,4)
70 NEXT

```

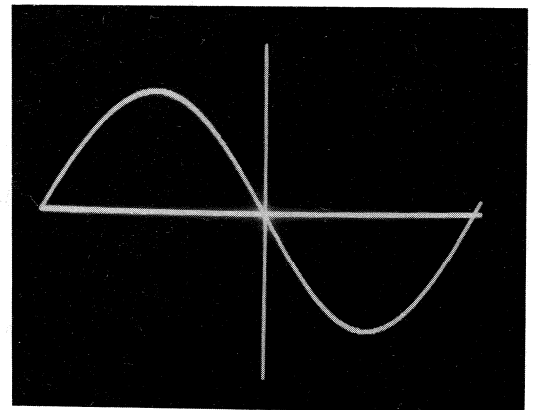


Photo 7.9 Sine Curve Drawing

This is a sine curve program using the SIN function. Lines 20 and 30 draw the co-ordinates. Numbers 40-70 calculate the position of the points and draw the curved line.

In the next example, we combine the SIN and COS functions to draw a circle -

```

10 CLS
20 PI=3.14159
30 LINE(0,99)-(639,99),PSET, 7
40 LINE(319,0)-(319,199), PSET, 7
50 FOR I=0 TO 360 STEP 2
60 Y=-50*SIN(2*PI/360*I)+99
70 X=130*COS(2*PI/360*I)+319
80 PSET(X,Y,4)
90 NEXT

```

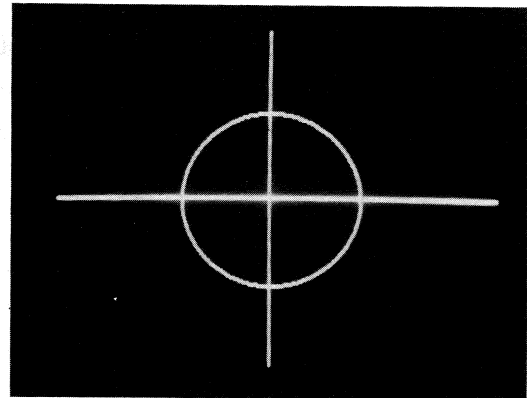


Photo 7.10 Drawing a Circle

Line numbers 30 and 40 draw the co-ordinates. Line 60 is the calculation for the direction of the Y axis and line 70 is for the X axis. The screen's vertical and horizontal ratio is not exactly 1 to 1. Therefore, to compensate for the distortion, we multiply by -50 and 130 in line numbers 60 and 70 respectively. This program draws a circle, but if you multiply the argument in the SIN function, you can show the X:1 resurge. Let's experiment by changing the numeric value.

```

60 Y=-50*SIN(2*2*PI/360*I)+99
      ↑
    multiple

```

You can add a B or BF option to the LINE statement. B stands for BOX and it enables us to draw a rectangle with the apexes of two opposite angles.

```
LINE(X1, Y1)-(X2, Y2), PSET, 7, B
```

If BF is designated, the inside of the rectangle is painted out with the designated colour.

```
LINE(X1, Y1)-(X2, Y2), PSET, 7, BF
```

```

10 CLS
20 X=RND(1)*640:Y=RND(1)*200
30 C=INT(RND(1)*7+1)
40 LINE -(X,Y),PSET,C,B
50 GOTO 20

```

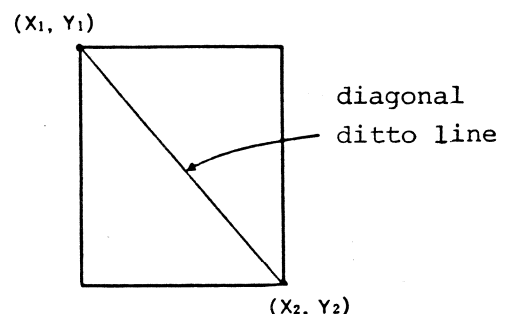


Fig. 7.11 Setting a Rectangle

If you change B in line 40 to BF, you can get various coloured rectangles.

Let's combine this with a LINE statement to make the program for a solid body.

```

10 CLS
20 X11=200:Y11=50
25 X12=350:Y12=100
30 X21=250:Y21=20
35 X22=400:Y22=70
40 LINE(X11,Y11)-(X12,Y12),PSET,4,B
50 LINE(X21,Y21)-(X22,Y22),PSET,4,B
60 LINE(X11,Y11)-(X21,Y21),PSET,4
70 LINE(X12,Y12)-(X22,Y22),PSET,4
80 LINE(X12,Y11)-(X22,Y21),PSET,4
90 LINE(X11,Y12)-(X21,Y22),PSET,4

```

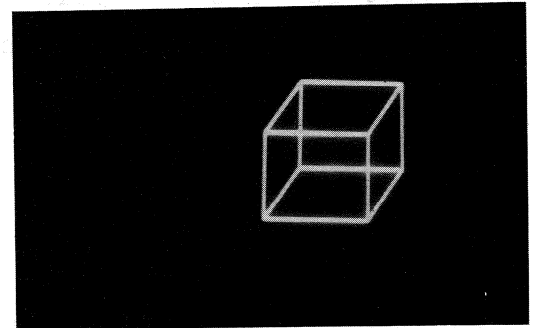


Photo 7.12 Drawing a Parallelepiped

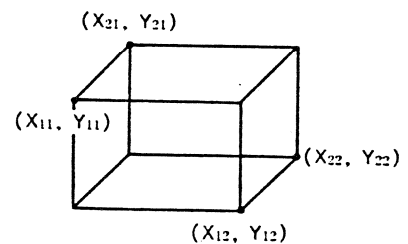


Fig. 7.13 Setting a Parallelepiped

A rectangular parallelepiped is shown on the screen. The relationship of the variables and the apexes is shown in Fig. 7.13.

Lines 20-35 contain the data for the apex positions. The shape of the cube would change with different values for X and Y.

* Using the LINE statement in the Character Mode

The LINE statement can be used in the character mode as well as in the graphic.

Designate two points and connect both points with the designated characters. This is not graphics so that characters are substituted for PSET and PRESET.

```

LINE(X1, Y1)-(X2, Y2), "A", C
      ↑           ↑           ↑           ↑
      starting  end   character colour
      point    point

```


The range of X_1 - X_2 and Y_1 - Y_2 is respectively 0-79, 0-24 for 80-ch. mode (WIDTH 80) and 0-39, 0-24 for 40-ch. mode (WIDTH 40). The character should be enclosed in ("). A character constant is also acceptable. B or BF may be used as options exactly the same as with the graphics.

```
LINE( $X_1$ ,  $Y_1$ )-( $X_2$ ,  $Y_2$ ), "A", C, B
                                or
                                BF
```

B displays the frame with the designated character and BF fills the inside with the designated character.

```
10 CLS
20 LINE(0,0)-(79,29),INKEY$,4,B
30 GOTO 20
```

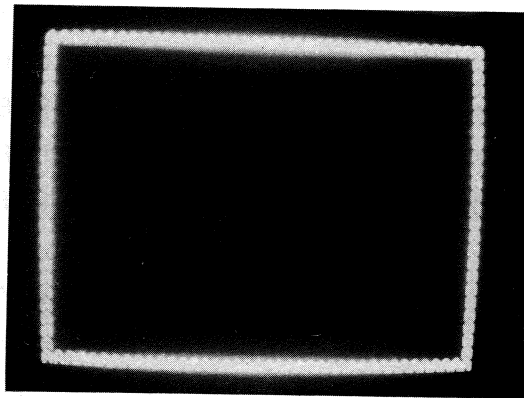


Photo 7.14. Example with **A**

INKEY\$ is not a variable but a function to take one character from the keyboard. It is a very short program but how does it work?

Key input is expected since we used INKEY\$. When we press a key, the screen is framed by the character corresponding to the input key (B designation). This character mode LINE command is quite useful in achieving a new effect in display.

* PAINT - Colouring figures

Painting a rectangle was simply accomplished by the BF option in the LINE statement. But, how do we go about painting in other figures?

Level-3 has a PAINT command for this purpose. By designating an arbitrary point inside the figure, the whole inside of the figure is painted the set colour with the PAINT command. Any figure, so long as it is a closed figure, may be painted

within its boundary line.

```
PAINT(X,Y), C, C2
           ↑   ↑
           |   |
paint colour boundary colour
```

As we do with PSET, we designate the position of the points with X and Y. C is the painted colour and C2 is the boundary colour. (C2 can be omitted.) There is only a colour resolution of 40 (no 80) so that the boundary colour cannot always be seen after painting out.

Let's execute this program -

```
10 CLS
20 LINE(100,50)-(200,80),PSET,4
30 LINE -(120,130),PSET,4
40 LINE -(100,50),PSET,4
50 PAINT(105,60),5
```

This draws a green rectangle and paints the inside blue. But if you look at the boundary lines of the rectangle, you will notice they are very difficult to distinguish. To make the boundary line clearer, we add C2. The line to be changed is number 50 -

```
50 PAINT(105,60),5,4
                   ↑
                addition
```

Take the LIST line number 50, move the cursor and add (,4). If you compare this result with the previous execution you will understand the meaning of C2. This program used blue to paint it but you can use the same green as is in the boundary line too.

So far, we have been using the Level-3 in the 640 x 200 high resolution mode. How do we designate the colour if in the 320 x 200, 160 x 100 or 80 x 100 mode?

For example, you may think that the designation of the argument would be 0-319 in the 320 x 200 mode. But this is not so.

We regard the LINE and PSET range as 0-639 whatever the mode. For the 320 x 200 mode, horizontally every second value expresses the position of the luminescent dot. This is the same as in 640 mode where two dots light together.

In 160 x 100 graphic mode, horizontally each fourth dot of the 0-639 range is expressed and vertically each second dot is expressed in the 0-199 range.

Similarly, in the 80 x 100 graphic mode, it is each eighth dot of 0-639 horizontally and each second dot of 0-199 vertically. Refer to Table 7.15 for this relationship.

Table 7.15 Numerical Values for Setting the Lighting Dots in each Mode

Mode	Horizontal Direction	Vertical Direction
640 x 200	0, 1, 2, 3, 4, ... 639	0, 1, 2, 3, 4, ... 199
320 x 200	0, 2, 4, 6, 8, ... 638	0, 1, 2, 3, 4, ... 199
160 x 100	0, 4, 8, 12, 16, . 636	0, 2, 4, 6, 8, ... 198
80 x 100	0, 8, 16, 24, 32, 632	0, 2, 4, 6, 8, ... 198

Now, what would happen if we designated the value of points other than the points which light in 160 x 100 mode?

Designate 1 where 0 should be designated. We execute PSET directly with PSET(1,0,7). A white dot is set at the 0,0 position. This shows that we can designate any of the four points from the 0-3 range of values. That is, the values to designate the points that light become a range of values - 0-3, 4-7 and 8-11. Try experimenting further for yourself to confirm this.

2. GRAPHIC CONTROL BY MACHINE LANGUAGE

* Level-3 VRAM Structure

The operation of graphics in Level-3 can be performed by many graphic commands. Sometimes, however, you may want to draw a pattern directly on the screen or read the screen without using a graphic command. It is necessary to understand hardware graphic structure to use machine language. So let's talk for a while on this subject.

We can regard the graphics on the screen as if they were the memory contents being displayed. This is readily understandable if we consider the VRAM (Video RAM).

We set the screen to 640 x 200 dots graphic mode. The total memory required to make one screen in this mode is calculated as follows -

As there are 640 dots for the horizontal direction -
 $640 \div 8 = 80$ (bytes).

There are 200 dots in the vertical direction. Therefore one screen is displayed by 80 (bytes) x 200. That is, the necessary memory to display the screen is $640 \div 8 \times 200 = 16,000$ (bytes) or 16 Kbytes. But, you will need more memory to use the dot image colour display.

Let us check the allocated VRAM capacity. Fig. 7.16 is the memory map.

VRAM is allocated 16 Kbytes from 0400-43FF. This is the maximum capacity for display use. This matches the total memory needed to display one screen in our above calculation.

In short, we know VRAM has a maximum of 640 x 200 dot graphic mode lighting dot information. VRAM does not have colour information.

* The structure of 5-bit colour information and colour register -

One unit of colour information had five bits. Its structure is shown in Table 7.17. (The ON and OFF in the table is the switch for whether that colour is output or not.) Bit 4 designates whether the VRAM content is character or graphic.

Bit 3 gives you normal or inverse information. In dot or character display, it lights if it is normal and the reverse happens, that is, the surrounding area lights if it is inverse.

normal A

inverse A

Bits 2-0 designate the colour. Bit 2 is green, bit 1 is red and bit 0 is blue. Each of the three primary colours is allocated its own bit. The seven colours Level-3 can display are displayed depending on combinations of these three bits as shown in Table 7.18.

Table 7.17 5-Bit Colour Information Structure

bit data	bit 4	bit 3	bit 2	bit 1	bit 0
0	character	normal	(green) G OFF	(red) R OFF	(blue) B OFF
1	graphic	inverse	G ON	R ON	B ON

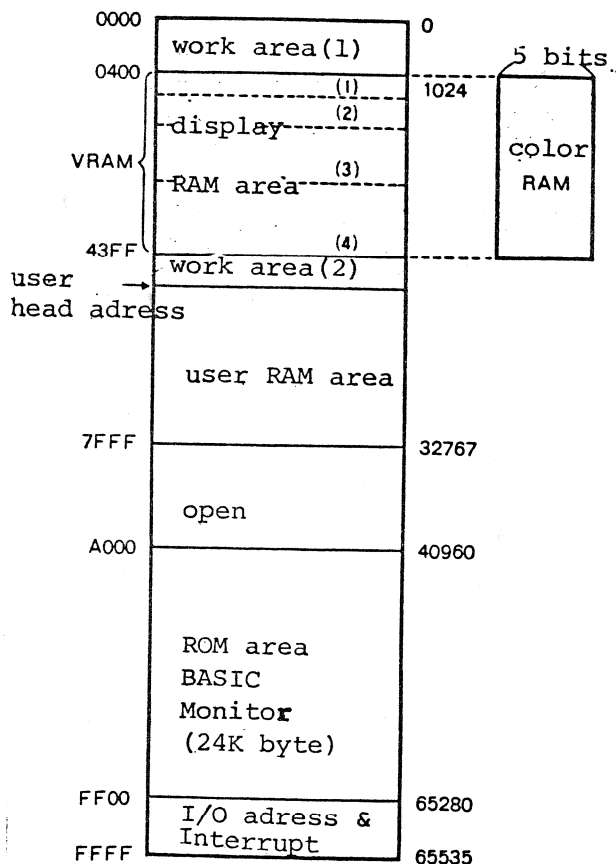


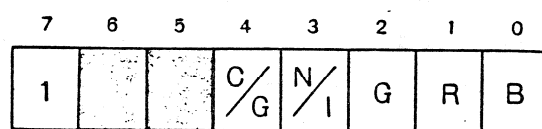
Fig. 7.16 Memory Map & VRAM

Table 7.18 Setting the Seven Colours with Bits 2-0

colour code	0	1	2	3	4	5	6	7
(green) bit 2	0	0	0	0	1	1	1	1
(red) bit 1	0	0	1	1	0	0	1	1
(blue) bit 0	0	1	0	1	0	1	0	1
colour	black	blue	red	magenta	green	cyan	yellow	white

Next we will talk about the construction of the colour register which is the gateway for output/input of the colour information. The colour register is allocated in address &HFFD8. The exchange of data located here may be thought of as only writing data in the memory and reading it from the memory.

The information to write in address &HFFD8 is 5 bits but to write this, we make bit 7 1 and make it an 8-bit address. The 8-bit register address &HFFD8 is shown in Fig. 7.19 below -



character normal
graphics inverse

Fig. 7.19 The Structure of Colour Register &HFFD8

You must always set the colour information into the register before writing the data in VRAM.

Let's examine the operation of the colour register. We have made a BASIC program here to understand the colour register control more easily.

```
10 CLS
20 SCREEN0
30 WIDTH 80
```

```
40 POKE &HFFD8,&H84
50 POKE &H0400,&H41
60 LOCATE 0, 10
```

When you execute this program, a green A character is displayed in the upper left corner.

Lines 10 - 30 include the screen's initial setting. In this case, 80-ch. normal mode is set. Line 40 sets the colour register and line 50 writes the data in VRAM. Line 60 resets the cursor position and protects the data from being destroyed.

We will explain this in more detail.

Look at line number 40. This is the command to write &H84 to address &HFFD8. If you convert this &H84 to a binary it becomes 10000100₍₂₎. If you apply this to the colour register shown in Fig. 7.19, we understand that this colour information is character, normal and green. Let's now look at line 50. This line designated &H41 to be written in address &H0400.

The head of VRAM is &H0400 and it is situated in the upper left corner of the screen. We know too that &H41 is the character A from the character code table. We can understand now how a green character A is displayed at the upper left corner of the screen.

Let's try displaying all the character code. Look at the next program. This is not a display program using the PRINT statement, but a program to write the characters directly to VRAM.

```
10 SCREEN0
20 WIDTH 40
30 FOR I=0 TO 255
40 POKE &HFFD8,&H84
50 POKE &H400+I,I
60 NEXT
```

```

70 FOR I=0 TO 255
80 POKE &HFFD8,&H89
90 POKE &H500+I,I
100 NEXT
110 FOR I=0 TO 255
120 POKE &HFFD8,&H92
130 POKE &H600+I,I
140 NEXT
150 LOCATE 0,20

```

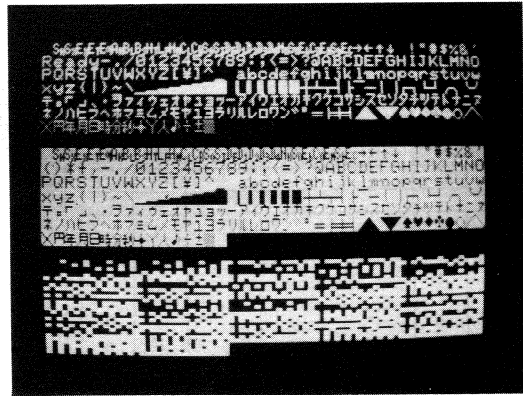


Photo 7.20 Example Program Execution
Using Colour Register

Line 10 and 20 set 40-ch. normal mode. Line 40 sets the colour information as character, normal and green. Line 80 and 120 set the command to input the colour information to the colour register. When you convert &H89 to binary code in line 80, it becomes $10001001_{(2)}$ which means character, inverse and blue.

Line 120 is graphic, normal and red because if you convert &H92 to binary code it becomes $10010010_{(2)}$. Therefore, when this program is executed, all the characters are displayed each in green, blue and red as shown in Photo 7.20.

* Screen Structure of High Resolution Mode

The above two programs are set at SCREEN 0, in other words - normal mode. How do we display SCREEN 1 high resolution mode?

The basic line of thinking is the same as in normal mode but it becomes a little more difficult because the high resolution mode VRAM structure is more complex than in normal mode.

The structure of VRAM at normal mode as shown in Figs. 7.21 and 7.22 is regarded as being continuous both logically and visually. But, with the VRAM structure in high resolution mode, the memory continuity and the visual continuity do not correspond.

40 ch.							
400	401	402	403	426	427	
428	429	42A	42B	44E	44F	
450	451	452	453	462	463	
⋮	⋮	⋮	⋮		⋮	⋮	
7C0	7C1	7C2	7C3	7E6	7E7	

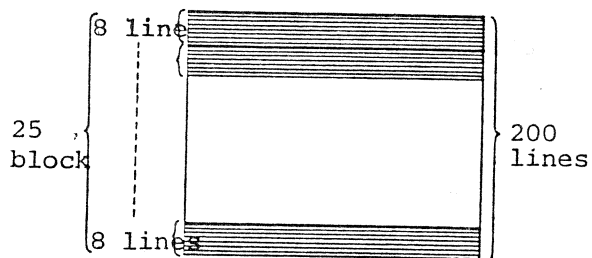
Fig. 7.21 Correspondence
Between Screen & Address
in 40-ch. Normal Mode
(Figures are Hexadecimals)

80 ch.							
400	401	402	403			44E	44F
450	451	452	453			49E	49F
4A0	4A1	4A2	4A3			4EE	4EF
⋮	⋮	⋮	⋮			⋮	⋮
880	881	882	883		BCE	BCF

Fig. 7.22 Correspondence
Between Screen & Address
in 80-ch. Normal Mode
(Figures in Hexadecimals)

With high resolution mode,
there are 40 bytes or 80
bytes horizontally and 200
lines vertically.

In Level-3, these 200 lines
are controlled in blocks of
25 as shown in Fig. 7.23.



7.23 High Resolution Mode
Screen Structure

Therefore one block consists of eight lines. The head address
of the block matches each line address at normal mode. Fig.
7.23 shows the top line of each block with a thick line.
For example, the 25 lines in 80-ch. normal mode correspond
with this thick line. Accordingly, the next line to follow
a certain line in the memory is the eighth line from that
and the first line of the 25th block connects to the second
line of the first block.

To explain this in more detail.

One line in high resolution mode has a memory capacity of
25 blocks which matches one screen in normal mode. Therefore
one high resolution mode screen is equal to eight screens in
normal resolution. That is, the necessary memory for the
screen in 320 x 200 mode (8 Kbyte) and 640 x 200 mode (16 K

byte) is eight times that of 40-ch. normal mode (1 Kbyte) and 80-ch. normal mode (2 Kbyte).

Therefore, the screen structure at 320 x 200 mode is as in Fig. 7.24 and 640 x 200 mode as in Fig. 7.25. If you look at the addresses in these figures the next address after the first line is obtained by adding either 1K ($1024_{(10)}$) or 2K ($2048_{(10)}$) to the first line address.

So, calculating the address to display graphics is not so difficult after all. For example, let's find the address of the position immediately below a certain position in 320 x 200 mode.

We do not have to do a 16-bit calculation by adding $0400_{(16)}$. Just divide the address into its upper address and lower address and do an 8-bit calculation adding $04_{(16)}$ to the upper bytes.

It is the same with 640 x 200 mode. We add $08_{(16)}$ to the upper bytes. Carrying out an 8-bit calculation makes the speed of the graphic display faster. The 6809 CPU has both a 16-bit and 8-bit addition command but the 8-bit addition is faster. We cannot completely ignore the execution time especially when we use high resolution mode because there are many more picture cells.

The head address of the 25 blocks of high resolution mode is the same as the address of the head of each line in normal mode. Thus, we can use this as the address base and can display colour at any position on the screen.

With high resolution, colour information is also given to the colour RAM by the colour register in the same manner as for normal mode.

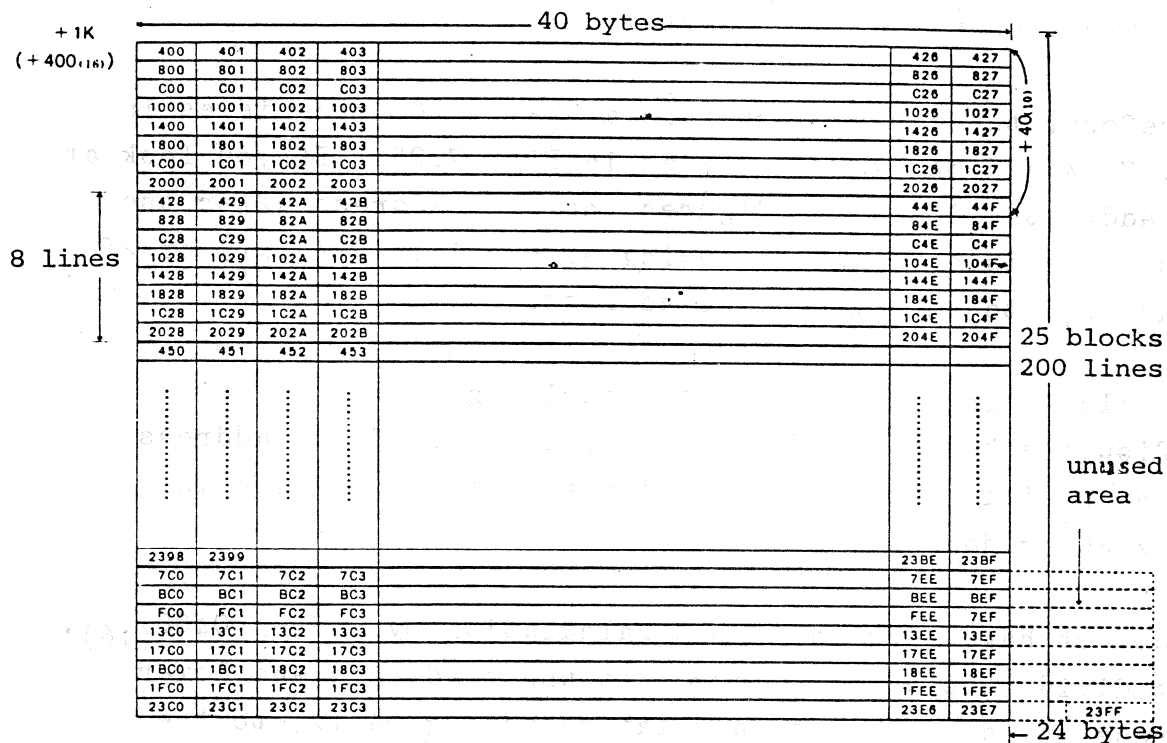


Fig. 7.24 320 x 200 High Resolution Mode Screen Structure
(Figures are hexadecimal)

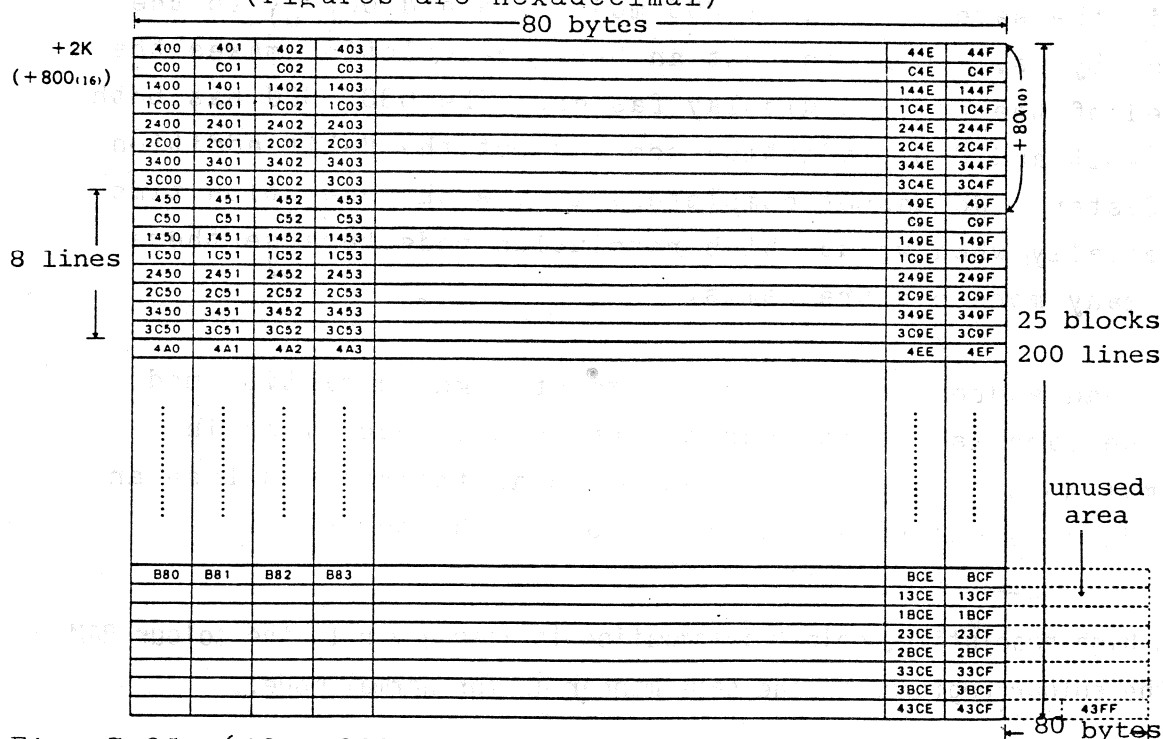


Fig. 7.25 640 x 200 High Resolution Mode Screen Structure
(Figures are hexadecimal)

* Colour Character Display in High Resolution Mode

How do we display characters in high resolution mode?

On a simple level, it is necessary to use graphics to make the characters since all the screen is in graphics. Still, in Level-3, you can display the characters on the screen by writing the character codes in VRAM as with normal mode. However, the character make-up is an 8-byte construction so that we write the character code in its 8 bytes.

Of course, we must set bit 4 in the colour register to 0 (character mode).

Let's display a green character A on the screen.

```
10 SCREEN1
20 WIDTH 80
30 FOR I=0 TO 7
40 POKE &HFFD8,&H84
50 POKE &H400+I*&H800,&H41
60 NEXT
70 LOCATE 0,10
```

The green character A is displayed in the left corner. In the above program, line number 40 writes the colour information in the colour register. Line number 50 covers the address calculation and writes the data in its memory. &H41 is the character code for A. This is written in VRAM for 8 bytes vertically by the FOR-NEXT loop. In this case, we can consider that the characters are sliced in the horizontal direction.

To digress, you may like to combine the inbuilt character pattern in Level-3 and display it. Horizontally you can write different character codes in eight lines but vertically this is not possible.

So much for the Level-3 VRAM structure.

Graphic control looks difficult at first but it can be done relatively easily if you understand the structure. The sample program is written in BASIC but a machine language program can be made with the same points and is much faster. Referring to the above, you may like to try.

Normally a machine language program is more compact and easier to use if it is made with BASIC. This is because in the initial setting and with complex calculations, BASIC is far simpler and more flexible. There is a method for transferring data from BASIC to machine language and it is also possible to make the program combining machine language and BASIC.

CHAPTER EIGHT

DISK BASIC MA-5300

In order to use floppy disk drives there are several commands to control them. We call these commands additional to regular BASIC, DISK BASIC. We will outline here the hardware needed to use disk BASIC and the procedure for starting it.

* Preparation

To use floppy disk drives we need -

- Mini floppy disk unit MP-3540
- Mini floppy disk card MP-1800
- Disk BASIC master diskette MA-5300 or MA-5301

If you have these ready, set the interface card MP-1800 in Level-3.

When you remove the cover, you can see six interface slots as shown in Fig. 8.1.

Insert the MP-1800 in one of these slots. It does not matter which one but since the card is a little big, perhaps I/F-1 would be best. Once inserted correctly, put the cover back on.

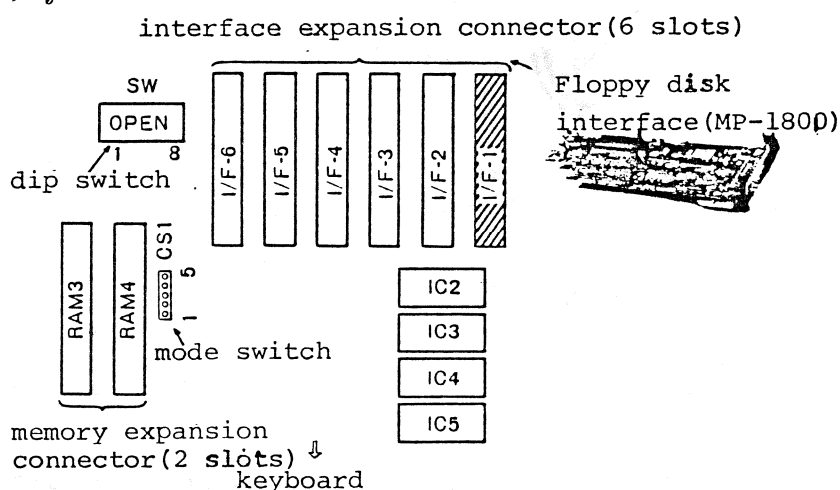


Fig. 8.1 Level-3 MPU Board

Next, connect the floppy disk drives to the set interface (MP-1800). The MP-1800 has a connector for this purpose and the floppy disk drive has the cable. The connector and cable can only be connected one way.

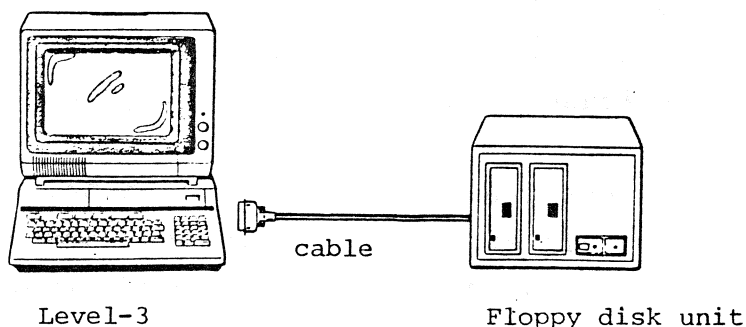


Fig. 8.2 Level-3 & Floppy Disk Connection

Thus connecting is simplified and incorrect connections are avoided.

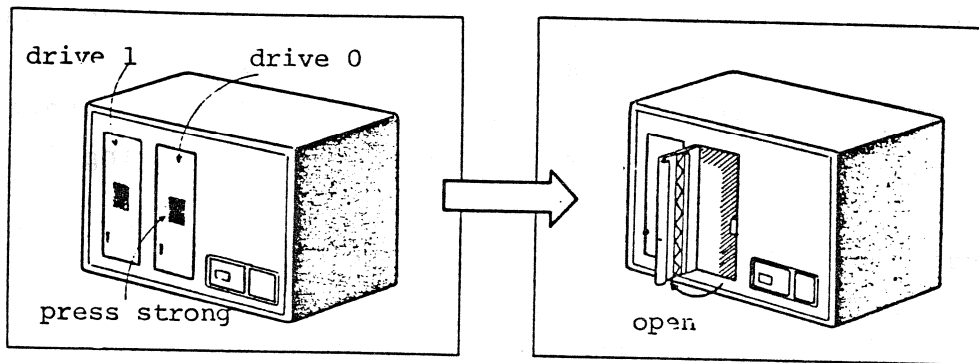


Fig. 8.3 Opening the Disk Drive

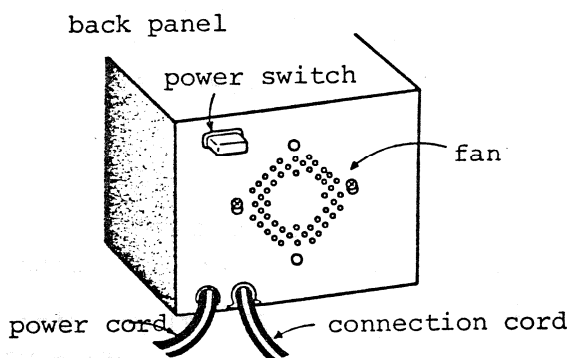


Fig. 8.4 Back View of the Floppy Disk

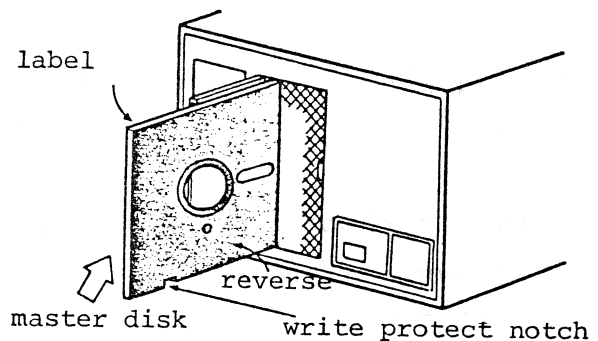


Fig. 8.5 Inserting the Master Diskette

* Start (Power On)

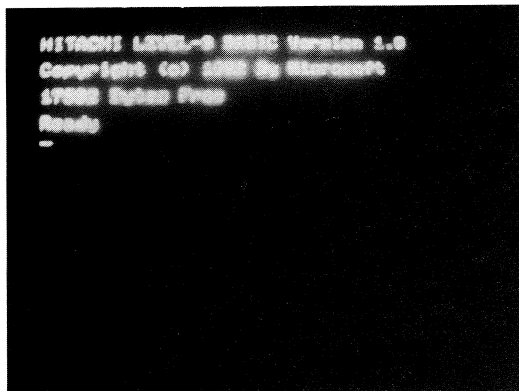
Let's now start disk BASIC - it is very easy.

First, turn the power on with the switch located on the back of the drive unit (see Fig. 8.4). Then open the disk drive panel cover 0. When you press the panel firmly, the lock is released and the cover springs open (see Fig. 8.3).

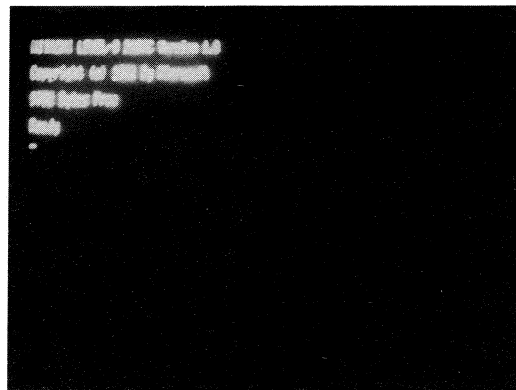
Insert the master diskette in this drive. The label side of the diskette should face the opened cover side (see Fig. 8.5). After inserting the diskette, close the cover, then switch

the power on the Level-3.

The Level-3 then automatically starts reading the disk and the 'Ready' condition is displayed on the screen after one or two seconds. Of course, the memory size you can use varies depending on whether the MODE switch is in the 0 or 1 position. If either of the two screens shown in Photo 8.6 appears, the machine is working normally.



(MODE 0)



(MODE 1)

Photo 8.6 Screen when Using Floppy Disk

* The Difference between Normal BASIC and Disk BASIC

The handling of the file descriptor is slightly different in disk BASIC. In normal BASIC, if the file descriptor is omitted, it signifies the cassette recorder, but in disk BASIC, it always indicates Drive 0.

That is to say, up to now if you had input LOAD, it would have displayed 'Searching' and read the program from the cassette tape. But in disk BASIC, disk is addressed. Therefore, in disk BASIC you must use LOAD "CAS 0" if you use the cassette recorder. Care is needed with this point.

To avoid trouble and have everything operating smoothly, you should get into the habit of always using a file descriptor when exchanging a program with outside peripherals.

The following commands are added in disk BASIC -

Table 8.7 File Descriptor

File	File Descriptor
Disk Drive 0	0:
Disk Drive 1	1:
Disk Drive 2	2:
Disk Drive 3	3:
Cassette	CAS 0:
Printer	LPT 0:
RS-232C	COM 0:
Screen	SCRN:
Keyboard	KYBD:

DSKINI Put the initialised diskette in usable state.

DSKO\$ Write the data on one sector of the disk.

FIELD Allocate the random buffer.

FILES Display the file name on the disk.

GET Read one record of the random file into random buffer.

KILL Delete one file on the disk.

LSET Input the data from program to random buffer from the left.

RSET Input the data from program to random buffer from the right.

NAME Change the file name.

PUT Write the record from random buffer to random file.

In disk BASIC the following functions are added too -

CVD Convert the 8-character argument into double precision numerical integer.

- CVI Convert the 4-character argument into single precision numerical value.
- DSKF Have the value of disk unused area.
- DSK\$ Read one sector's contents on the disk.
- LOC Give the accessed file number before this function's execution.
- LOF Have the maximum record number in random file as the value.
- MKD\$ Convert the double precision numerical value argument into 8-byte character string.
- MKI\$ Convert the integer argument into 2-byte character string.
- MKS\$ Convert the single precision numerical value argument to 4-byte character string.

* Usable Memory Capacity

The memory capacity usable by disk BASIC is about 4K bytes smaller than cassette base BASIC. This is because in order to use the disk, the disk control program recorded in the master diskette and additional command program are loaded into the memory.

Table 8.8 Usable Memory Capacity with Disk BASIC

Mode	Capacity (Byte)
40-ch. Normal	25112
80-ch. Normal	24088
40-ch. High Res.	17944
80-ch. High Res.	9752

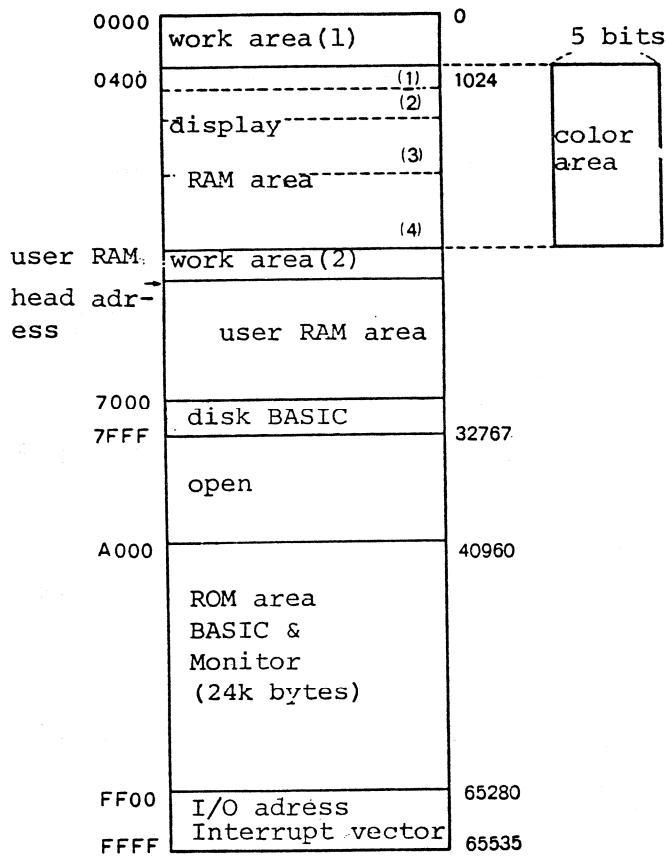


Fig. 8.9 Memory Area in Disk BASIC

CHAPTER NINEKEYGET

* Input Method without using Return Key

When we input the character from the keyboard, under normal conditions, we have to press RETURN. However it becomes inconvenient if you have to press RETURN each time, such as in a game. In Level-3 we have the INKEY\$ for this purpose.

If INKEY\$ is executed and a key is pressed, its character becomes the value of the function. If no key is pressed during the execution, null character becomes the value of the function. With this, we can omit the need to press RETURN when we input.

```
10 A$=INKEY$
20 IF A$="" THEN 10
30 PRINT A$;
40 GOTO 10
```

This is an example of a simple program using INKEY\$. INKEY\$ is used as a substitution sentence as in line 10. When line number 10 is executed, the characters of any keys pressed are substituted for A\$. Then line 20 judges the contents of A\$. (") means null character. Line numbers 10-20 form a loop and thus the same program is repeated until characters are set in A\$ by INKEY\$.

When this program is RUN, no cursor appears, but when you press the appropriate keys their corresponding characters appear on the screen. From this, we can confirm that characters are input without using RETURN and the program is executed.

To make it a little more interesting, we will make a program which shows the cursor position and which uses cursor control keys and the insert/delete key.

The point of this program is to have the cursor flicker and to input characters. All the commands except the SCREEN function, INKEY\$, POS(0) and CSRLIN are familiar to you.

```
10 SCREEN0
20 X=0
30 Y=CSRLIN
40 A=SCREEN(X,Y)
50 IF A=0 THEN A=32
60 LOCATE X,Y
70 COLOR 15
80 PRINT " ";
90 COLOR 7
100 FOR I=0 TO 30
110 A$=INKEY$
120 IF A$="" THEN NEXT ELSE 190
130 LOCATE X,Y
140 PRINT CHR$(A);
150 FOR I=0 TO 30
160 A$=INKEY$
170 IF A$="" THEN NEXT ELSE 190
180 GOTO 30
190 LOCATE X,Y
200 PRINT CHR$(A);
210 IF A$=CHR$(13) THEN PRINT:GOTO 250
220 LOCATE X,Y
230 PRINT A$;
240 IF A$=CHR$(18) THEN 30
250 X=POS(0)
260 GOTO 30
```

Line 20 sets the initial horizontal value and line 30 sets the initial vertical value. The SCREEN function (line 40) has the value of the X,Y position character code. The space becomes 0.

The cursor character must flicker on the screen. To do this, the character is incorporated in the variables with the SCREEN function.

Line number 50's IF statement sets the space character code to A only when A is 0. Lines 30-90 write the cursor character on the screen and its flicker is controlled in lines 30-180.

Look at the flow chart as the program is a little hard to understand (Fig. 9.1 on following page).

To make input possible while the cursor is flickering the following is required.

- (1) Reserve the characters on the screen.
- (2) Write the cursor.
- (3) Watch the key input.
- (4) If no key input, return the characters on the screen.
- (5) Watch the key input.
- (6) If no key input, go to (1).
- (7) If there is input, write its character on the screen.

Let these points sink in and then look at the flow chart again.

There is a large loop and we leave this loop from two places. These are the key inputs (that is, the procedure for characters being taken by INKEY\$).

At the two input points, the FOR-NEXT loops are related to the cursor flicker time. As the number of loops increases the flicker time also increases.

Normally we think in the order of:

- writing the character
- waiting period
- input

But with this, the response to the input becomes slow in proportion to the waiting time. To overcome this, INKEY\$ is placed in the FOR-NEXT loop and we can go out from the loop any time once there is input.

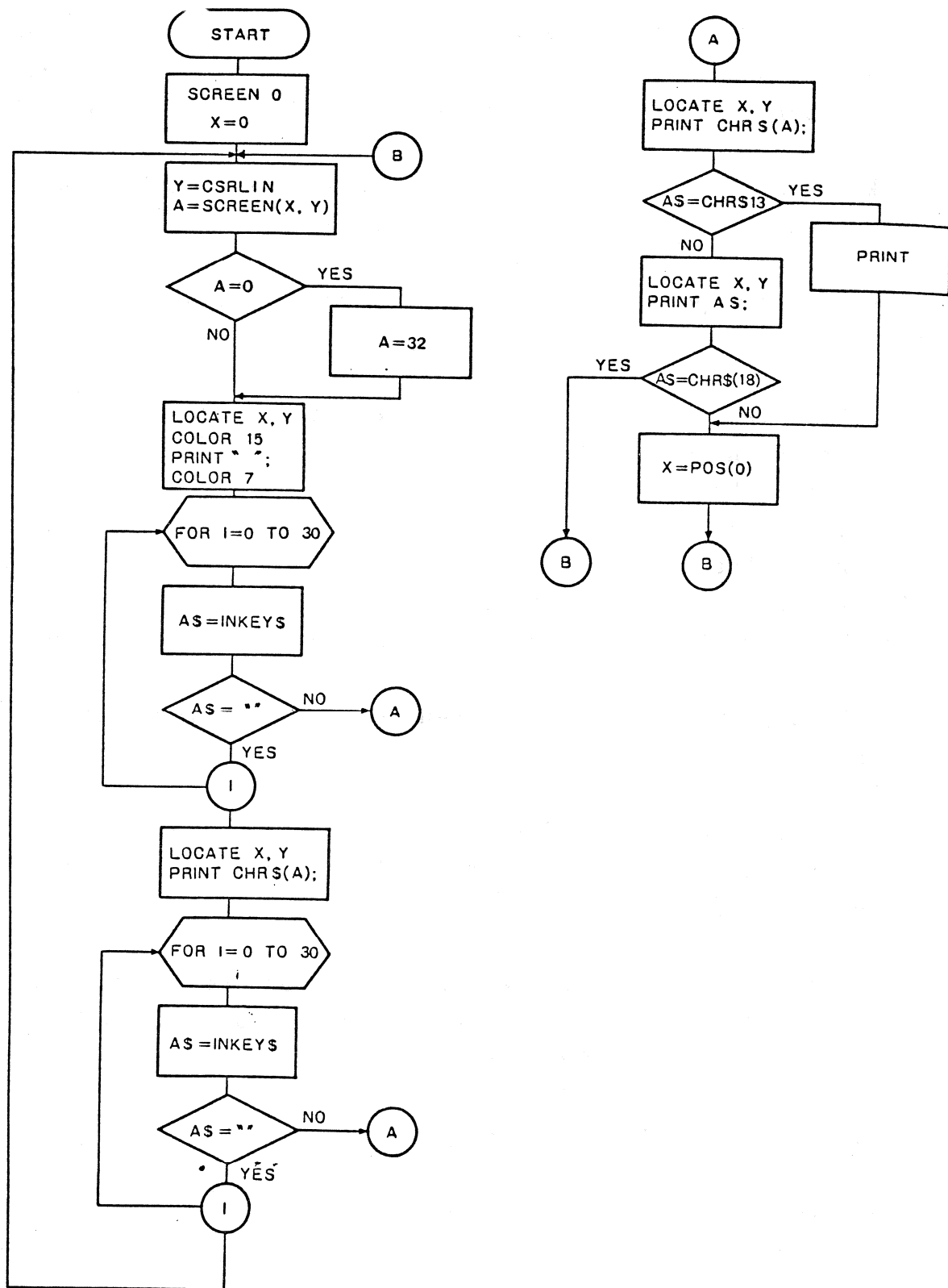


Fig. 9.1 Flow Chart for Inputting with using the Return Key

Line number 190 on governs the procedure for input. `INS` and `RETURN` require a special procedure.

At line 230, A\$ input by INKEY\$ is output but as this PRINT statement has (;) a new line is not started.

The RETURN code is a decimal 13 so if this is output straight, the cursor returns to the head of the line with no subsequent line feed.

At line 210, when RETURN is detected, output only PRINT is output and then a line feed is made. Line 250 sets the cursor's horizontal position to X.

The INS code is 18. At line 240, if the PRINTed character is CHR\$(18), X is not renewed and we return to key input. The 'Insert' is to move the characters after the cursor position. The cursor does not move and so the value of X is not renewed.

When you RUN this program, the cursor flickers. Press the appropriate keys and the relevant characters are displayed on the screen. Of course the cursor control keys, `INS`, `DEL`, `CLS`, `HOME` and other control keys may be used without any syntax error occurring.

* Further Applications

We generally use an INPUT statement when inputting character strings in a commercial program or a game program.

When the machine has been placed in input mode with an INPUT statement, it accepts cursor key input and `HOME` `CLS` etc., so we can inadvertently erase what is on the screen. Especially when we input numbers with the numerical TEN keys, the risk of mis-operation becomes high because the cursor keys are near these keys.

To avoid this and to make the input not more than a specified number of characters, we must prepare an input program which is equivalent to the INPUT statement by ourselves.

Therefore, let us change the previous input program with cursor control and give it the equivalent function of INPUT.

Preparation to set input character buffer and input/output condition -

- (1) Input buffer is one-dimensional array and the array name is KB\$(n).
- (2) Set the variable BE to control the input character number and to go to a subroutine.
- (3) Return from the subroutine when the RETURN is pressed.
- (4) Input characters are compressed in FB\$ and we return to the main program. Note the return code is not included.
- (5) In subroutine, only the ordinary keys, RETURN and DEL are accepted.

Keeping these conditions in mind, let's change the previous program. The new program is on the next page.

Note that we have put REM statements in important places. The subroutine which we have as our objective runs from line 1000 on.

To cope with the LOCATE statement, we use the POS(0) and CSRLIN functions. With these two functions, we can make the cursor blink and input at any point on the screen.

Also when the program is in the subroutine, the buffer pointer KP and substitute variable KB\$ are initialised (line 1000-1020).

```

10 ' main program (sample)
20 CLEAR 500:'<----- set strings buff
30 DIM KB$(255):'<-- init key buff
40 BE=40:'<----- set buff length
50 GOSUB 1000:'<---- call input subroutine
60 PRINT KB$:'<----- KB$:input strings
70 GOTO 50
1000 ' subroutine input
1010 X=POS(0):KP=0:KB$="":' KP is a buff pointer
1020 Y=CSRLIN:A=SCREEN(X,Y)
1030 IF A=0 THEN A=32
1040 LOCATE X,Y:PRINT"*";:' cursor character
1050 FOR I=0 TO 30
1060 A$=INKEY$:' get one character
1070 IF A$="" THEN NEXT ELSE 1130
1080 LOCATE X,Y:PRINT CHR$(A);
1090 FOR I=0 TO 30
1100 A$=INKEY$:' get one character
1110 IF A$="" THEN NEXT ELSE 1130
1120 GOTO 1020
1130 LOCATE X,Y:PRINT CHR$(A);
1140 IF A$=CHR$(13) THEN PRINT:GOTO 1230:' check return code
1150 IF A$=CHR$(8) THEN KP=KP-1:LOCATE X,Y:PRINT A$;:GOTO 1210
:' check DEL code
1160 C=ASC(A$)
1170 IF C<32 THEN 1020:' cancel control code
1180 KB$(KP)=A$:KP=KP+1:' store input character and increment
pointer
1190 IF BE<=KP THEN BEEP:KP=KP-1:X=POS(0)-1:GOTO 1020:' check
buff length
1200 LOCATE X,Y:PRINT A$;
1210 X=POS(0)
1220 GOTO 1020
1230 FOR I=0 TO KP-1:' set KB$ from KB$(n)
1240 KB$=KB$+KB$(I)
1250 NEXT
1260 RETURN:' return from subroutine

```

In this program, '*' is designated in the cursor position but as you see in line number 1040, it is designated by the PRINT statement. Thus, it can be exchanged freely. If you want to change the cursor colour, put a COLOR command before or after this statement.

Line number 1140 checks the RETURN key and line number 1150 checks the DEL key. With the DEL key, we should not increase the pointer so we put KP=KP-1. For the same reason, in line 1190 we have an Input Buffer Over judging statement. Line 1170 cancels the control code. All the control code is allocated after the space and nothing happens under 32 (ASCII decimal space code) except for the DEL and RETURN keys.

Line 1230 and after is the routine for compressing the

characters in the array to KB\$. It is a simple FOR-NEXT loop so you will be able to understand it easily.

(How to Use the Subroutine)

Set the input buffer at the head of the program (DIM KB\$(255)). Then input the number of characters in BE if input character control is operative. If you do not limit it, set BE=255 at the beginning.

In this subroutine, we use GOSUB 1000. As this is the same as INPUT KB\$, it will perform the same as if you had changed all the INPUT statements to GOSUB 1000.

* Keyget for Games

In Level-3, dual input is possible. That is, when the computer is executing one job, you can input other key input.

However, it is inconvenient to use this function sometimes, for example, when executing a game program.

The biggest problem is that characters accumulate in the key buffer because of the repeat functions. In BASIC, the INKEY\$ command which is often used in real time games, has the function of reading the characters in the buffer in order. Because of this, the key input and the game's progress become inconsistent.

For this kind of situation, we use a function key which does not repeat. We use the key which corresponds to the character used in the program.

When you make a real time game in BASIC, consider various program orders of execution and design the fastest execution possible. A slow execution cuts the fun of the game in half.

The knack to making a good game is twofold - 'how fast you move the pattern' and 'how fast you make the key input'.

Let's look at the program -

```

10 SCREEN 0,,1
20 P$="<-o->"
30 B$=" "
40 KEY 1,CHR$(29):KEY 2,CHR$(28)
50 X=40:Y=20
60 LOCATE X,Y:PRINT P$
70 A$=INKEY$
80 IF A$=CHR$(29) THEN 120
90 IF A$=CHR$(28) THEN 160
100 IF A$=" " THEN 200
110 GOTO 70
120 LOCATE X,Y:PRINT B$
130 X=X-1
140 LOCATE X,Y:PRINT P$
150 GOTO 70
160 LOCATE X,Y:PRINT B$
170 X=X+1
180 LOCATE X,Y:PRINT P$
190 GOTO 70
200 LOCATE X+2,Y:PRINT " "
210 K=X+2
220 FOR I=Y TO 0 STEP -1
230 LOCATE K,I:PRINT "^"
240 LOCATE K,I:PRINT " "
250 NEXT
260 LOCATE K,Y:PRINT "o"
270 GOTO 70

```



Line numbers 70-110 form the key input routine. Here we use the INKEY\$ function and get each character from the keys. The input character is analysed by the IF statement and branches off to its respective program.

The same effect is obtained by using the INPUT\$ function. At the beginning of the program, open the file with -

```
OPEN "I",#1, "KYBD:"
```

and change the INKEY\$ to -

```
A$=INPUTS(1, #1)
```

The cursor is displayed then. Line number 60 designates erasing the cursor with LOCATE.

```
LOCATE X, Y, 1 : ---
```

However, with these two methods, there is no repeat but if many keys are pressed, they are stored in the buffer and the result is the same as before.

The last way is to prohibit keyboard interruption and to read the keys directly. The actual address is unknown at present but from our experience, address &HFFEO seems to be the clue. We have found that by writing &HOC in this address, we can see the characters without interrupt. However, the key can only be read by the cursor control keys or the space key.

```
10 POKE &HFFEO,&HOC
20 PRINT PEEK(&HFFEO);
30 GOTO 20
```

Try inputting this program and pressing RUN. You should be able to see many one-column numbers. If you press a normal key there is no change. What will happen if you press a cursor key or space key?

The numbers stop at that point. Let's use this instead of keyget.

The space is 128, the upper arrow is 129, the left arrow sign 131, the down arrow sign 132 and the right arrow sign is 133. Let's input these in our program and revise it as follows -

```
40 POKE&HFFEO,&HOC
70 A=PEEK(&HFFEO)
80 IF A=131 THEN 120
90 IF A=133 THEN 160
100 IF A=128 THEN 200
```

then add this line -

```
5 POKE &HFFEO,&HOC
```

Press the RESET button to stop the execution. You could stop the program with `BREAK` too but it then will not accept key input as it is in the prohibit key interrupt condition. The only way is to use RESET and return the keys to their original state. When you return the system with the program, write `POKE&HFFEO,&HFO`.

If you are making a real time game, it is recommended to read direct with `PEEK`.

CHAPTER TENGENERATING SOUNDS* Generating Sounds with BASIC

As you know, the Level-3 has a built-in speaker which generates sound. For example, when you press a key, a click sound will be heard and if you make a mistake, a slightly unpleasant beep can be heard.

Also in BASIC, we have a BEEP statement which generates the same sound as happens with an error. What do we need to do to make this sound more pleasant?

The indistinct sound is caused by intrusion from the timer. Let's try stopping the timer -

```
POKE &HFFD4,255
```

The sound changes to a high single tone. But as the timer is stopped, we have lost the clock function. In general, when PRINT TIME\$ is executed, the renewed time is displayed but after a POKE statement execution, no matter how many times PRINT TIME\$ is executed, the displayed time stays the same and does not change. To disable this function, that is to recover the clock function, execute -

```
POKE &HFFD4,0
```

To make a slightly different sound, we use the address &HFFD3 which is connected to the speaker.

Of the eight bits in &HFFD3, actually only bit 7 is connected to the speaker, so we access this bit. There is a flip flop and an AMP between &HFFD3 and the speaker. If 0 or 255 is input to the flip flop, its state changes and, in this way, we can make a pulse. (See Fig. 10.1.)

By outputting these pulses, we can make sound. The pitch of the sound is decided by the pulse width and the pulse width is decided by the timing of the 0 and 255 input information

into the flip flop.

As BASIC has long execution times, we cannot generate very high sounds. To do this, we must use machine language. Of course, if you are not a real sound buff then BASIC will probably be quite sufficient.

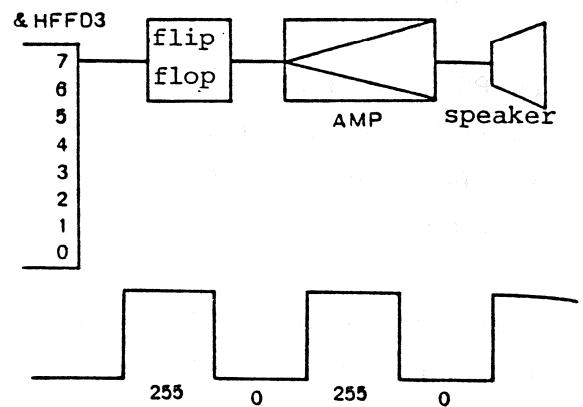


Fig. 10.1 Address &HFFD3 and Speaker, Pulse

Now let's use BASIC to generate some sound -

```
10 FOR I=0 TO 20
20 POKE &HFFD3,0
30 POKE &HFFD3,255
40 NEXT
```

Line 10 and 40 decide the length of the sound. Line 20 and 30 make the pulse by inputting 0 and 255 to &HFFD3. The speed of the process governs the maximum pitch of the sound. We cannot really say that the sound produced by this program is a pleasant one so we will change it a little more in the next program -

```
10 POKE &HFFD3,0
20 FOR I=0 TO 4:NEXT
30 POKE &HFFD3,255
40 FOR I=1 TO 4:NEXT
50 GOTO 10
```

Lines 20 and 40 set the waiting time. If executed, we get a lower sound than before. Let's erase line 40 and change line 20 to -

```
20 FOR I=1 TO 40:NEXT
```

In line 20 we only have to add 0 after 4, so move the cursor

to that position and insert it. The program now is -

```
10 POKE &HFFD3,0
20 FOR I=1 TO 40:NEXT
30 POKE &HFFD3,255
50 GOTO 10
```

When this program is executed, we get a staccato beeping sound (as opposed to an unbroken beeping). To change the sound even more -

```
10 FOR I=0 TO 4
20 POKE &HFFD3,0
30 POKE &HFFD3,255
40 NEXT
50 FOR I=0 TO 2
60 POKE &HFFD3,0
70 FOR J=0 TO 3:NEXT
80 POKE &HFFD3,255
90 NEXT
100 GOTO 10
```

and we get a different sound again.

So in BASIC, we can generate all sorts of interesting sounds with the program.

* Generating High Tones with Machine Language

To achieve the higher tones, we must execute the program faster. Unfortunately, BASIC is limited in how fast the program may be executed. To overcome this problem, we make a subroutine with a machine language program. We can make a machine language program using the same philosophy as the BASIC program. Fig. 10.2 gives the flow chart. This shows only the main procedure however and we cannot write the machine language into this flow chart.

A brief outline of this flow chart follows -

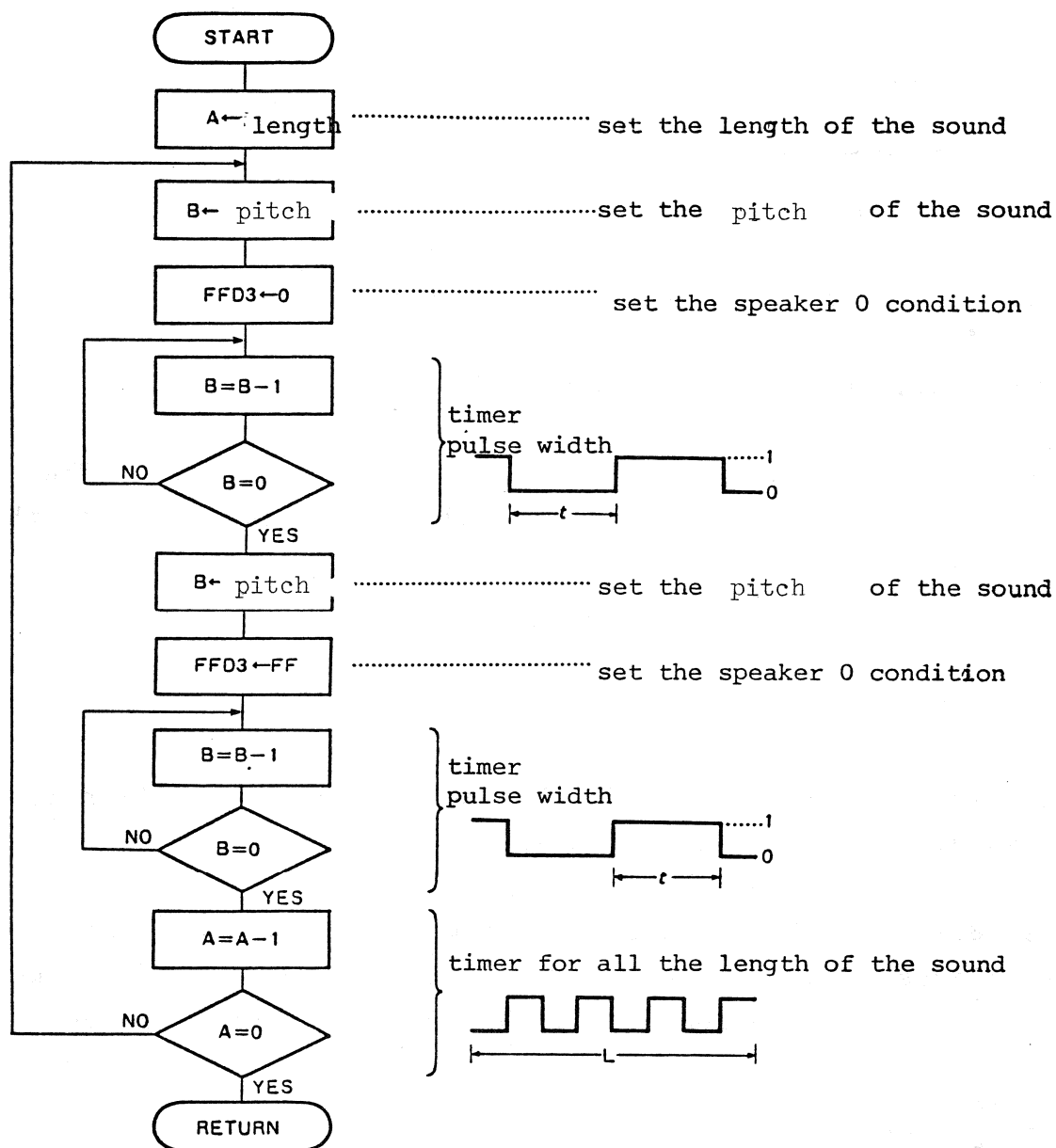


Fig. 10.2 Machine Language Program to Produce Sound - Flow Chart

The registers A and B are for tentative use and have nothing to do with the 6809 internal registers.

Firstly we set the length of one sound in A and then the data for its pitch in B. This pitch data is of a semi-cycle length and one cycle means both the 0 and 1 states have each been completed.

The pitch of the sound depends on the number of the cycles (that is, the frequency) in one second. The higher the

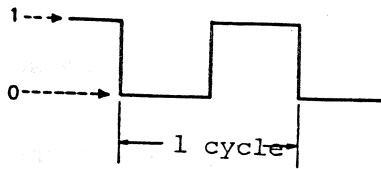


Fig. 10.3 Length of one Cycle of Data

frequency, the higher the sound.

The length of each state - 0 and 1 is $\frac{\text{cycle}}{2}$ (sec.). The cycle T (sec.) and frequency F (Hz) decides the pitch of the sound as follows -

$$F = \frac{1}{T} \text{ (Hz)}$$

The cycle T (sec) is decided by the number of the timer loops as shown in Fig. 10.2 flow chart.

The 6809 execution time for one command is on the average a few microseconds ($\mu S = 10^{-6} S$) so if we consider the loop time L (μS) (loop execution time), the frequency can be given as -

$$F = \frac{10^6}{L} \text{ (Hz)}$$

With this method, we can obtain the frequency from the loop time. The loop time is relatively accurate so we may even produce a musical scale. Table 10.4 shows the scale and the corresponding frequencies. Let's determine the loop time and actually generate some sounds.

The frequency ratio of a sound and that sound one octave higher is 1:2. If we can determine the basic octave, we can also get the other octaves with some simple calculation.

Let's find the loop time needed to generate 'doh'. According to Table 10.4 on the following page, the 'doh' sound is 261.63 Hz

so -

Table 10.4 Scale and Frequency

$$261.63 = \frac{1}{T}$$

$$T = 0.0038221 \text{ (sec.)}$$

This is the time for one cycle and so the loop time which corresponds to a half cycle is -

$$\frac{T}{2} = 0.001911 \text{ (sec.)}$$

The minimum loop program execution time in 6809 is about $5 \mu\text{S}$ so we can obtain the number of the loops as follows -

$$\frac{0.001911}{5 \times 10^{-6}} = 382.2 \text{ (loops)}$$

We now can see that we should generate 382 loops. The normal formula to get the number of the loops C is -

$$\text{No. of loops } C = \frac{1}{2 \times 5 \times 10^{-6} \times F} \text{ (loops)}$$

Note that this number is the number of half cycle loops. Now, the number of the loops above is 382 which exceeds the limit of the one-byte memory procedure (presently 256). Therefore, we must make a 2-byte loop program. However, if we raise the sound to be generated one octave higher, the frequency doubles and the cycle becomes $\frac{1}{2}$, so that the number of loops also becomes $\frac{1}{2}$ and we can thus easily store the data in one byte.

Following, we have made a simplified program to generate doh. The fundamental structure follows the flow chart in Fig. 10.2.

scale		frequency
do	C	261.63
	C #	277.18
re	D	293.66
	D #	311.13
mi	E	329.63
fa	F	349.23
	F #	369.99
so	G	392.00
	G #	415.30
ra	A	440.00
	A #	466.16
ti	B	493.88

	LDA	#n	- Set length of sound (n is sound length data address)
LN	LDB	#\$00	} Speaker → "0"
	STB	\$FFD3	
	LDB	\$n'	} Half cycle loop (n is loop data address)
LP1	DECB		
	BNE	LP1	
	LDB	#\$FF	} Speaker → "1"
	STB	\$FFD3	
	LDB	\$n'	} Half cycle loop (n is loop data address)
LP2	DECB		
	BNE	LP2	
	DECA		} Sound length count
	BNE	LN	
	RTS		- Return → BASIC

This program is a simple machine language program to generate some sound. We use EXEC command from BASIC and the sound length data and the scale data (no. of loops) as input. The n and n' used in the program are the sound length data (this is one byte) and the scale data memory address which is a 2-byte integer. This means the buffer can give the parameters to the subroutine. Therefore we have to POKE the two pieces of data into the memory and EXEC for the program to be executed.

The program does not work as it is however, and we must change to code. We can convert the mnemonic level program into hexadecimal code by looking at the code table.

Now we set n as \$6000, n' as \$6001 and start the program from \$6002. The OP code column in Table 10.5 is the machine code. you may compare it with the mnemonic column on the right. To input this machine code, we use the BASIC MON (MONitor) command.

MON RETURN

*

↑ cursor

Table 10.5 Machine Language Program to Produce Sound

adress	op code	label	numoric
6002	B6 60 00		LDA \$6000
6005	C6 00	LN	LDB #\$00
6007	F7 FF D3		STB \$FFD3
600A	F6 60 01		LDB \$6001
600D	5A	LP1	DECB
600E	26 FD		BNE LP1
6010	C6 FF		LDB #\$FF
6012	F7 FF D3		STB \$FFD3
6015	F6 60 01		LDB \$6001
6018	5A	LP2	DECB
6019	26 FD		BNE LP2
601B	4A		DECA
601C	26 E7		BNE LN
601E	39		RTS

The monitor command is executed using D (dump) and M (memory input). However, if we were to return to BASIC at this condition, depending on the variables, we could destroy the machine language program. To avoid this, we must institute memory control to reserve the machine language area.

First with -

CLEAR 300,&H5FFF

set the upper limit of the memory to &H5FFF and release the address &H6000 and on from BASIC.

Then execute the MON command and input the machine language program.

MON

*M6002

6002 00—

↑
cursor

With the M command, we set M6002 and input the machine code from &H6002. Once we finish putting them in from B6 to 39,

operate BREAK, or CTRL + C, and return to BASIC.

We have now the preparation for generating sounds. Let's try it. We set the sound data in &H6000 and the scale data to &H6001.

```
10 POKE &H6000,255      ← Sound length data set
20 POKE &H6001,60       ← Scale data set
30 EXEC &H6002          ← Execution
RUN
```

Key in this program and execute it. Now you will hear higher tones than we were able to with BASIC. Try changing the scale data in &H6001 and execute. This subroutine is useful for generating many effective sounds.

CHAPTER ELEVEN

USING THE LIGHT PEN

In Level-3, the light pen interface is incorporated as a standard so the Hitachi light pen can be fitted easily. The Level-3 BASIC has a light pen command and program preparation is easily accomplished. We can make use of the PEN function or interrupt.

* PEN function

We have mentioned the PEN function before but now we will explain in more detail.

The detect capability of the light pen is 40-ch. or 80-ch. horizontally and 25 lines vertically. The PEN function shows six different conditions with the argument (0-5). As will be outlined in the following section, in the trigger sense and level sense conditions, the computer detects whether the pen is triggered and then determines where the pen intersects the horizontal and vertical co-ordinates.

(1) Trigger Sense

The trigger sense condition detects the light pen position each time it is pressed on the screen. PEN(0) - PEN (2) form the trigger sense group.

PEN(0) becomes 0 when the pen is not pressed on the screen and -1 when it is.

PEN(1) and PEN(2) set the horizontal and vertical co-ordinates when the light pen is pressed on the screen (that is when PEN(0) is -1). Once determined the same value is kept until the light pen is pressed the next time.

(2) Level Sense

The level sense condition outputs the present position of the pen while it is held to the screen.

PEN(3) - PEN (5) make up the level sense group functions. They correspond to the PEN(0) - PEN(2) trigger sense group.

PEN(3) becomes -1 if the light pen is being held on the screen. If not, it becomes 0. PEN(4) and PEN(5) have the value of the position's co-ordinates when the pen is being pressed on the screen. However when the light pen is removed from the screen, it keeps the value it had just prior to being removed.

Here we have a trigger sense program -

```
10 WIDTH80:SCREEN1
20 COLOR0,7
30 GOSUB100:X1=X*8:Y1=Y*8
40 GOSUB100:X2=X*8:Y2=Y*8
50 GOSUB100:X3=X*8:Y3=Y*8
60 LINE(X1,Y1)-(X2,Y2),PSET,1
70 LINE-(X3,Y3),PSET,1
80 LINE-(X1,Y1),PSET,1
90 GOTO 30
100 IF PEN(0)=0 THEN 100
110 X=PEN(1):Y=PEN(2)
120 RETURN
```

Now to execute the program. If you press three points, we get a triangle with these three points as vertices. This program designates the background white in line 20. The light pen detects the position of the light, and white is quite easy for it to detect. The light pen is most sensitive to blue, reacts less easily to red and has no reaction to black at all.

For a level sense program -

```
10 WIDTH 80:SCREEN1
20 COLOR0,7:LINE(0,0)-(80,25),PSET,1,B
30 IF PEN(3)=0 THEN 30
40 X=PEN(4):Y=PEN(5)
```

```

50 LOCATE X,Y:PRINT" ";:REM GRAPH+L
60 PSET(X,Y,1)
70 GOTO 30

```

When you press the light pen and slide it over the screen, the pen draws over these positions touched. Furthermore, in 640 x 200 graphic mode, a similar figure to the one drawn is copied in the upper left corner.

The background colour is designated blue to which the light pen is most sensitive.

* Light Pen Interrupt

Another way to use the light pen is to use the interrupt. When the interrupt is used we do not read the co-ordinates as with the PEN function but we can designate multiple areas on the screen and judge in which area the light pen is pressed.

To set these areas, a PEN statement is prepared and its format is -

PEN $n_1:(X_{11},Y_{11})-(X_{12},Y_{12}),n_2:(X_{21},Y_{21})-(X_{22},Y_{22}), \dots$

n stands for the area number to distinguish between each area. We can set up to nine areas. If n is 0, it is regarded as an unspecified area.

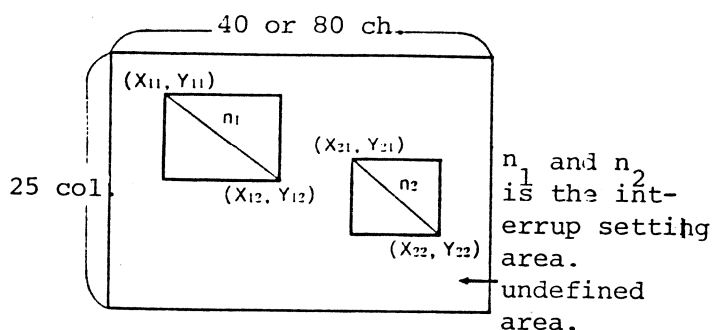


Fig. 11.1 Designating Areas with the PEN Statement

The set area is a rectangle designated by the diagonal line between the vertices $(X_{11},Y_{11})-(X_{12},Y_{12}), (X_{21},Y_{21})-(X_{22},Y_{22})$.


```

30 LINE(15,0)-(24,5),"⌘",10,BF
40 LINE(30,0)-(39,5),"⌘",12,BF
50 PEN 1;(1,0)-(9,5),2;(15,0)-(24,5),3;(30,0)-(39,5)
60 ON PEN GOSUB 100,200,300,400
70 PEN ON
80 GOTO 80:REM Endless loop
100 PEN OFF:LOCATE 0,20:PRINT"Error"
110 FOR I=0TO300:NEXT:LOCATE0,20:PRINTCHR$(26):PEN ON:RETURN
200 PEN OFF:LOCATE 0,20:PRINT"color:blue":GOTO110
300 PEN OFF:LOCATE 0,20:PRINT"color:red":GOTO110
400 PEN OFF:LOCATE 0,20:PRINT"color:green":GOTO110

```

Line number 50 sets three areas. Line 60 has the branch to go to the interrupt routine from line 100 on. If the light pen touches on an undefined area we go to line 100 and if one of the set area numbers of 1, 2 and 3 is pressed, we go to line number 200, 300 or 400.

Line 80 with GOTO 80 is an endless loop. According to this, assuming that the program is executing a procedure unrelated to the light pen, the light pen input is controlled.

Now, to execute this program.

Three rectangles - blue, red and green are drawn. When you press the light pen to the rectangle on the right, "color: green" is displayed on the bottom left. This shows that interrupt is accepted correctly despite the endless loop in line 80 being executed.

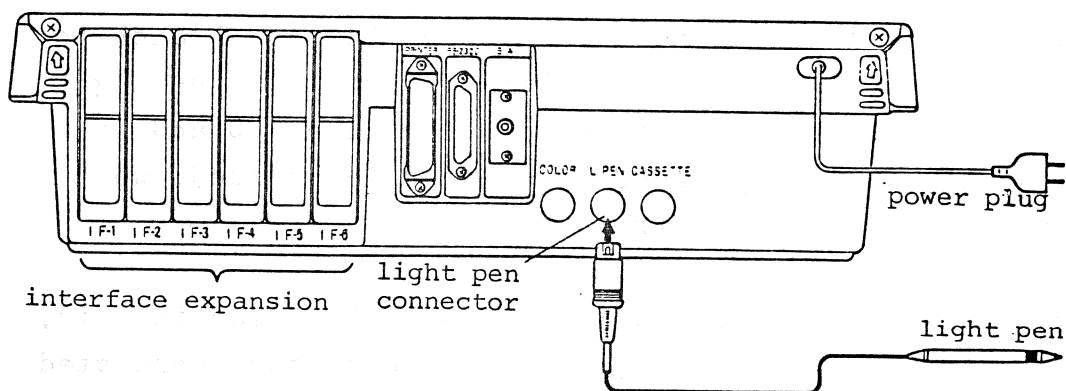
If you use this light pen, it gives you the feeling that you are really making use of the computer. For example, you can put your knowledge to practical use on such things as program menus and games.

* Light Pen Interface Connector

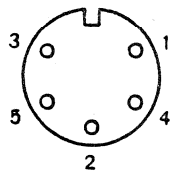
The light pen interface connector consists of five pins as shown in Diag. 11.3 and its signals are in Table 11.2.

Table 11.2 Light Pen Interface Connector

1	$\overline{\text{LP OUT}}$	Photodiode output voltage in the light pen
2	$\overline{\text{LP SW}}$	Signal showing that the light pen is pressed
3	+ 5V	+5V power supply
4	LP VCMP	Standard voltage to decide the light pen's sensitivity
5	GND	Ground



Diag. 11.3 Back Panel of the Level-3



Diag. 11.4 Light Pen Connection Connector Pin Arrangement

When the pen is pressed, $\overline{\text{LP SW}}$ becomes low level and $\overline{\text{LP OUT}}$, which shows the timing, is output.

The co-ordinates where the light pen is pressed are set in the CRT controller LSI and we read this information in the program. This controller chip manual is obtainable locally (HD46505SP).

CHAPTER TWELVE

MASTERING CHARACTER STRING CALCULATION

The data handled by BASIC is essentially numbers and characters. In the case of numbers, we can do simple four-rule arithmetic to complicated function calculation.

With character strings, it is not just a matter of displaying and comparing the data. We can also do some more creative jobs.

* Addition of Character Strings

Adding character strings means connecting the character strings to each other. It is different to the addition of numbers but the same symbol "+" is used.

"AB" + "CD" → "ABCD"

The calculation of character strings differs from numbers in that we do not have subtraction, division or multiplication. Be that as it may, each character string function is able to carry out complicated operations and these have been organised in Table 12.1.

Table 12.1 Function Table of the Character Functions

Input	INKEY\$ INPUT\$	Take one character from key Input from input buffer
Conversion	CHR\$	Convert from character code to character
	ASC	Convert character to character code
	VAL	Convert numeral to numerical value
	STR\$	Convert numerical value to numeral
	HEX\$/OCT\$	Convert numerical value to hexadecimal, octal character string
Process	LEFT\$	Cut character string from left
	RIGHT\$	Cut character string from right
	MID\$	Cut a part of character string

Information	LEN	Length of character string
	INSTR	Included in character string or not
Generation	STRING\$	Generate character strings
	SPACE\$	Generate space character strings

* Character Functions

Out of this table, the most commonly used character string operations are -

CHR\$, ASC, VAL, STR\$, LEFT\$, RIGHT\$, MID\$, LEN

Let's study these functions first to enable us to use them without any worry.

The Level-3 has a clock function, (If you want to see the time, we PRINT TIME\$ and it is displayed.) Let's display the time using the non-interlace mode characters (hours. minutes.seconds).

```

10 SCREEN0,,1:WIDTH 40
20 HH$=LEFT$(TIME$,2)
30 MM$=MID$(TIME$,4,2)
40 SS$=RIGHT$(TIME$,2)
50 LOCATE 10,10
60 PRINT HH$;"hours";MM$;"minutes";SS$;"seconds"
70 GOTO 20

```

Line 10 changes the character generator to non-interlace mode. HH\$ in line 20 is the data variable for (hour) and the substitution sentence to input two characters from the left to HH\$. We have used the LEFT\$ function.

Next, MM\$ in line 30 is the (minute) data variable and substitutes two characters four places from the left of TIME\$ for MM\$. SS\$ in line 40 is the (second) data variable and substitutes two characters from the right for SS\$.

Line 50 designates the display position and line 60 displays the time as (hour.minute.second).

From this example, you can understand that we can separate the character string using character functions.

* Data Compression and Resolution

This method is used by character functions to store data in the memory or in floppy disks.

Data of the same length can be stored in a small volume by compressing them with variables. We need a compression and resolution program. It is worthwhile trying it as it is not such a long project.

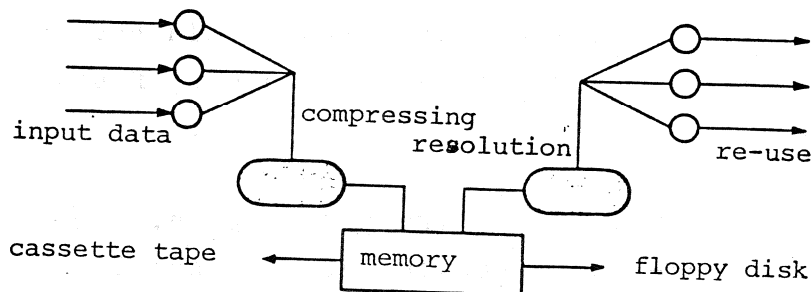


Fig. 12.2 Data Compression.Resolution Process

*Compressing the Data and Storing it

Particularly with programs of commercial applications it is very important to know how to store data most efficiently as there is a vast amount of data to be stored. Let's consider this next example. We memorise the ledger in the memory. We assume the necessary data is debits, credits, items, dates and amounts. We must also take care that if they are memorised separately, we may not be able to organise them if there is too much data or if excess memory is used.

We compress the data as shown in Fig. 12.3 and memorise it all as one piece of data. We can express this data in 18 characters.

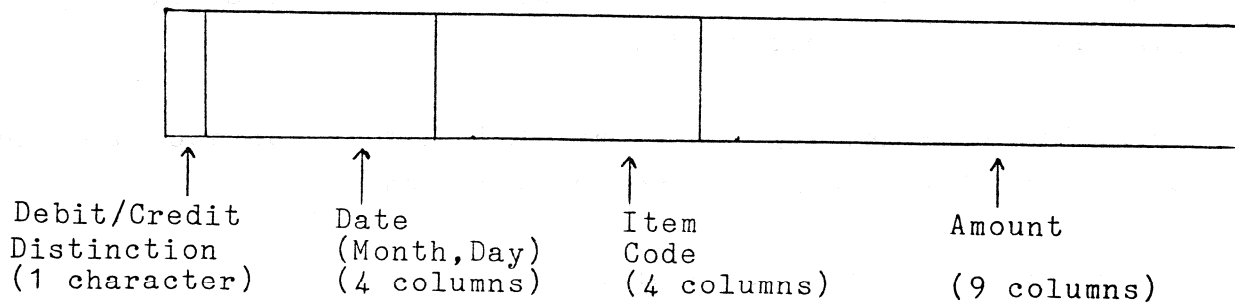


Fig. 12.3 Data Construction

Now for an actual program compressing data. We will only make it as a sample so we will only execute PRINT and will not write the data into file this time.

With this type of program, we must be particularly careful with the error procedure. An error in this case does not mean an error message. For example, it might be that we carry out character input where number input should have been effected, or, input five characters where only four-character input can be accepted.

As this sort of thing naturally is likely to happen, from the outset we need to take countermeasures. The sample program has a repeat function so that if unexpected input takes place, it starts again from the beginning. (You will need a more stringent check if actually used in a commercial situation.)

```

10 ' sample program
20 ' initialize
30 SCREEN0:WIDTH 40
40 ER$=CHR$(30)+CHR$(5)
50 ' input data
60 INPUT"MONTH:";M$
70 IF VAL(M$)<=0 OR VAL(M$)>12 THEN PRINT ER$;:GOTO 60
80 INPUT"DAY:";D$
90 IF VAL(D$)<=0 OR VAL(D$)>31 THEN PRINT ER$;:GOTO 80
100 INPUT"DEBIT:1 / CREDIT:2 ",K$

```

```

110 IF K$="1" OR K$="2" THEN 130
120 PRINT ER$;:GOTO 100
130 INPUT"ITEM CODE:",CD$
140 IF LEN(CD$)<> 4 THEN PRINT ER$;:GOTO 130
150 INPUT "AMOUNT:",KN$
160 IF KN$="" OR LEN(KN$)>9 THEN PRINT ER$;:GOTO 150
170 ' data packing
180 HI$=RIGHT$("0"+M$,2)+RIGHT$("0"+D$,2)
190 KK$=RIGHT$(SPACE$(9)+KN$,9)
200 DA$=K$+HI$+CD$+KK$
210 PRINT:PRINT"DATA:";DA$

```

Line 40 substitutes character codes 30 and 5 into ER\$. CHR\$(30) is the cursor control key ↑ and CHR\$(5) is CTRL + E.

When ER\$ is output by the PRINT statement, the cursor position moves up one line and executes CTRL + E. If there is incorrect input, the input is once more requested at the initial position and the screen is not effected. This job is carried out by the IF statement which comes after each INPUT statement.

The data is compressed from line 180 on.

HI\$ is the variable to input data with dates. There are four characters secured for the date (two characters for the month and two for the day). You may notice the tricks in using the RIGHT\$ function. When you input, it is a little unnatural to push the keys 01 for January, for example. To make two characters from a one-character input we do -

```
RIGHT$("0")+M$,2)
```

In M\$, both one-character and two-character numbers have been entered for the month data. Putting 0 at the head of the number means, for example, that January is 01 and December is 012. With RIGHT\$, two characters are taken from the right, so if it is one character, the 0 stays there at the head of the number and if there are two characters the

number is shown as it is (without the 0). The data for the date is added in the same way so HI\$ is always four characters no matter what month or day.

There are nine character spaces reserved for the amount and we correct the input numbers to nine characters. In the same way the date was treated, we put a suitable character at the head of the number and then take nine characters from the right.

```
RIGHT$(SPACE$(9)+KN$,9)
```

or

```
RIGHT$(STRING$(9,48)+KN$,9)
```

When a space is required at the head of the number, we use the SPACE\$ function. Add KN\$ to this and count off nine characters from the right. If 0 is required, generate nine 0s by the STRING\$(9,48) and add KN\$ to it. An alternative way would be to write nine 0s, "000000000"+KN\$.

In line 190 nine characters are set in KK\$. Up to now it has been the preparation for the compressed data. In the next line 200, the added data is set in DA\$.

As shown in this program, we want to use the data as efficiently as possible.

* Using the Compressed Data after Resolving it

We now need to make a program to resolve the compressed data so it can be used. We can resolve the data quite easily as the data structure is decided.

```
10 ' sample
```

```
20 DA$="203222945 12345"
```

```
30 K$=LEFT$(DA$,1)
```

```
40 HI$=MID$(DA$,2,4)
```

```

50 CD$=MID$(DA$,6,4)
60 KK$=RIGHT$(DA$,9)
70 '      print sample
80 MS$="DEBIT:":IF K$="2" THEN MS$="CREDIT"
90 PRINT"DEBIT/CREDIT: ";MS$
100 M$=LEFT$(HI$,2):M=VAL(M$)
110 D$=RIGHT$(HI$,2):D=VAL(D$)
120 PRINT "DATE : "M;"MONTH";D;"DAY"
130 PRINT "ITEM : ";CD$
140 KK$=VAL(KK$):PRINT USING"AMOUNT:###,###,###":KK$

```

Line number 20 sets certain data. Lines 30-60 resolve the compressed data. A distinguish code for debit/credit is set in K\$. Likewise the date for HI\$, an item code in CD\$, and amount in KK\$ are also set.

By using LEFT\$, RIGHT\$ and MID\$ we can sort the data out easily.

This program can be used to bring compressed data from elsewhere, totalise it and make tables and so on. It is useful if you make such a program into a subroutine.

* Making Data Base Software

Judging whether a character is included in a certain variable is performed by the INSTR function. This function has been mentioned earlier. Here we will use it to make simple data base software.

A data base is a system for structuring information and housing it so we can put various data in and take it out again as necessary. If you want to obtain some information you can get it if you know the keyword.

For the personal computer, we are not able to devise overly complicated ones but we can make a procedure that will amply meet your own needs. A simple data base program is written on the following pages.

```

10 ' sample program (data base).
20 CLEAR 2500:SCREEN0:WIDTH80
30 DIM T$(100):P=0
40 FD$="SCRN:"
50 PRINT"*** easy data base ***":PRINT
60 INPUT">",C$
70 IF C$="I" THEN 210
80 IF C$="E" THEN 700
90 IF C$="D" THEN 760
100 IF C$="S" THEN 370
110 IF C$="L" THEN 280
120 IF C$="L/" THEN 670
130 IF C$="P" THEN 460
140 IF C$="P/" THEN 900

150 IF C$="T" THEN 870
160 IF C$="CLR" THEN 840
170 IF C$="SAVE" THEN 500
180 IF C$="LOAD" THEN 590
190 IF C$="HELP" THEN 950
200 GOTO 60
210 ' input
220 PRINT"input mode"
230 LINEINPUT L$
240 IF L$="Q" THEN 60
250 T$(P)=L$
260 P=P+1
270 GOTO 230
280 ' list
290 OPEN"O",#1,FD$
300 B=0
310 FOR I=B TO P-1
320 PRINT#1,USING"####";I;:PRINT#1," ":T$(I)
330 IF INKEY$=CHR$(27) THEN CLOSE:GOTO60
340 NEXT
350 CLOSE
360 GOTO 60
370 ' search
380 INPUT"S:",D$
390 OPEN"O",#1,FD$
400 FOR I=0 TO P
410 IF INSTR(T$(I),D$)<>0 THEN PRINT#1,T$(I)
420 IF INKEY$=CHR$(27) THEN CLOSE:GOTO 60
430 NEXT
440 CLOSE
450 GOTO 60
460 IF FD$="SCRN:" THEN FD$="LPT0:":PRINT"printer mode":GOTO 60
470 FD$="SCRN:"
480 PRINT"screen mode"
490 GOTO 60
500 ' data save
510 INPUT"SAVE:file name",FL$
520 X$="CAS0:"+FL$:OPEN"O",#1,X$
530 PRINT#1,P
540 FOR I=0 TO P
550 PRINT#1,T$(I)
560 NEXT
570 CLOSE
580 GOTO 60
590 ' data load
600 OPEN"I",#1,"CAS0:"
610 INPUT#1,P
620 FOR I=0 TO P
630 INPUT#1,T$(I)
640 NEXT
650 CLOSE
660 GOTO 60
670 INPUT"from:",B
680 OPEN"O",#1,FD$
690 GOTO 310
700 ' edit
710 INPUT"Edit No.",L

```

```
720 PRINT T$(L)
730 PRINT CHR$(30);
740 LINEINPUT T$(L)
750 GOTO 60
760 ' delete
770 INPUT"DEL No.",L
780 FOR I=L TO P-1
790 T$(I)=T$(I+1)
800 NEXT
810 P=P-1
820 GOTO 60
830 ' clear
840 P=0
850 PRINT"text clear"
860 GOTO 60
870 ' pointer print
880 PRINT"pointer: ";P-1
890 GOTO 60
900 ' mode print
910 M$="screen"
920 IF FD$="LPT0:" THEN M$="printer"
930 PRINT"mode: ";M$
940 GOTO 60
950 ' help
960 PRINT:PRINT"Instructions":PRINT
970 PRINT"( 1) I    input text (Q) is escape"
980 PRINT"( 2) L    list    (ESC) is stop"
990 PRINT"( 3) L/   list from N line"
1000 PRINT"( 4) E    edit"
1010 PRINT"( 5) D    delete"
1020 PRINT"( 6) S    search"
1030 PRINT"( 7) P    mode switch(priter/screen)"
1040 PRINT"( 8) P/   P/S mode display"
1050 PRINT"( 9) T    pointer display"
1060 PRINT"(10) SAVE text save"
1070 PRINT"(11) LOAD text load"
1080 PRINT"(12) CLR  clear text buff":PRINT
1090 GOTO 60
```

With a large data base for general use, we normally work on the data structure and sometimes use the hardware to make the data search easier. However, in this program, we will only display the line containing the object character string.

Now to explain the program. Please refer to the pertinent line number as you read.

First, there is the input command analysing part and the IF statements decide the branch of the program.

Command input prompt statement is (>) in line 60. It is annoying if (?) appears in the waiting condition, so we use (,) in the INPUT statements. Therefore, with the INPUT statement in line 60, only (>) is displayed when input is requested.

The commands which are set here are as follows -

I	input	L	list	L/	list from line n
E	edit	D	delete	P	printer switch
SAVE	record	LOAD	read	S	search
P/	mode	CLR	delete	T	pointer
	confirmation		all		confirmation

They each have an IF statement which has its subroutine so that the computer comes back to the INPUT statement in line 60 once the routine is finished.

If the program is executed, first (>) is displayed and we go into the input waiting condition. Then we input one of the commands mentioned above.

I command -

If I is input, the computer goes into the character string input mode. The cursor is blinking so input any character string you like. To return from this mode to command input mode, press @ , then hit RETURN.

The variable P in line number 250 is used as the pointer to show the order of character strings. In this program, we do not use insert to insert character strings between lines. Thus T is used to show which line character strings are in.

L command, L/ command -

L command has the same function as the BASIC LIST command. If L is input, the computer outputs the data from the head line in the character string. If you want to pause temporarily, press CTRL + S. This is originally Level-3's function. If you want to stop in the middle, press ESC.

In Level-3, normally we use CTRL + C to stop but if this is pressed, the whole program comes to a halt. We therefore press ESC instead. Whether ESC is pressed or not is decided by INKEY\$ in the next IF statement (27 is the character code for ESC).

```
IF INKEY$=CHR$(27) THEN 60
```

We input L/ to see the character strings from the nth line. "from" is displayed and if, for example, you want to see from the seventh line, input 7. The other functions of L/ are exactly the same as L. If you examine the program you will understand that both these commands use the same subroutine.

P command -

P has the switch function to change from printer to screen and vice versa.

When you input P, the file descriptor changes to LPT 0 for the printer and SCRIN for the screen and outputs "Printer Mode" or "Screen Mode" correspondingly.

If the file is opened with the OPEN command, we can use it with the file descriptor in the variable and so we do not

have to make the same program for the printer if we have the contents in the variable for the equivalent file descriptor (in this case LPT 0:).

E command -

E stands for edit and is defined to correct the input line. When E is input, the edit number and line number we should correct appear on the screen.

Input the number output during the L command then that line is displayed and the cursor moves to the head of this line. Correct it, move the cursor to the last character of the line, then hit RETURN. As we use LINE INPUT for this, if you press RETURN in the middle of the line, only the character strings from the head of the line to that particular position would be displayed.

D command -

D stands for delete. If D is input, the DEL No. and the line to be deleted are displayed. As we did with the E command, input the line output by the L command. In the program, the D command subroutine starts from line 760. The input data is all contained in the T\$() string, so that if you want to delete some lines in the middle, you must move the lines that follow forward. This job is performed in the FOR-NEXT loop in lines 780-800.

T command -

The T command prints where the pointer is at the present moment. Simply output the contents of P.

S command -

S stands for the search function. When you input S, "S:" is displayed and the computer asks for the character string you want searched.

For example, say there is an address list in the data base and the character string you want searched is a certain person's address and name. In this case, if you input the name after S:, his name and address is output.

Now look at line 370 and on. The character strings input with S: have been entered in variable D\$. Then, the INSTR function in line 410 judges whether the character string is included here. If the character string in D\$ is not included in the character string in T\$(I), the value of the INSTR function is 0. If included, it is not 0 and so the program goes to the line number designated by THEN and displays all the character strings.

CLR -

When CLR is input, all the input character strings are deleted. The CLR operation is only a matter of resetting the pointer P to 0.

So much for our explanation of the program. Try adding a routine to insert between the lines in this simple data base. As well as D command, move the data in the array and renew pointer P once. You can modify this program as you wish by adding other functions.

CHAPTER 13USING THE PRINTER

* Connecting the Printer to Level-3

When you debug long programs or save calculation results or issue simple dockets it is most convenient to have a printer.

Level-3 has a centronics interface so you can connect up a printer compatible with this standard with just one cable.

Most current printers have the same pin connection so there should not be any problems but if you are using a printer not specifically for Level-3, check the pin arrangement to make sure.

When you print with the printer, the data is once stored in the printer buffer. And with the data 'Print', all the data in the buffer is printed. This resembles the time when you input the program and press RETURN and execute the first input.

The code to instruct 'Print' is generally called the control code. Level-3 control code is L/F (line feed) &HOA as a standard and this is sent to the printer. In other words, by sending &HOA, we advise the end of one line and printing is commenced.

The control code varies on the printer, but typical ones included in Level-3 are shown in Table 13.1.

Table 13.1 Representative Control Codes

Control Code	Character Code
L/F	&HOA
C/R	&HOD
C/R + L/F	&HOA + &HOA

We close the printer file with -

```
CLOSE #n
      ↑
    file number
```

'#' can be omitted here also. The file number must be the same number designated by the OPEN. If you omit this number n, all the open files are closed.

```
OPEN "0", 1, "LPT0:"
OPEN "0", #3, "LPT0:"
CLOSE 1
CLOSE #2
CLOSE 1, 2, 3
CLOSE
```

* PRINT #n, PRINT #n, USING - Output to the printer

We use PRINT #n when we output the results of calculations of characters and numerical values. The method is the same as with the PRINT statement and #n is the file number designated by OPEN.

To print a particular print format, use PRINT #n, USING which has the same function as PRINT USING. Tables can easily be written up with PRINT #n, USING. Let's make a program -

```
10 OPEN"0",#1,"LPT0:"
20 FOR I=1 TO 10
30 PRINT#1,I;
40 NEXT
50 PRINT#1
60 CLOSE
```

When you execute this program, the printer prints out the numbers 1-10.

The last ';' in line 30 means continue so we do not send control code here.

The printer stores up the data in its internal buffer continuously. Then, in line 50 the print command is given to the printer by PRINT#1. Without this line, data is only stored in the buffer and does not start printing. You must be careful to understand this relationship between the print command and the control code or a mistake will result.

As in the above program, if you add ';', do not forget to start a new line with PRINT#n.

To confirm this, delete line number 50 and execute the program. The printer will not print. Then execute PRINT#1 directly and the printer starts to print 1-10.

With the printer, ';' requires attention.

* Drawing on the Printer's Various Functions

Recently printers are equipped with microprocessors with a lot of functions. For example, the normal 80 characters can be made double size or reduced to that approximately 130 characters can be printed, or even graphics may be printed.

The functions are designed for various fairly complex applications. However, to use these functions, we must send the anticipated code from Level-3 to the printer. These codes do not correspond with the characters. One of &H00-&H1F codes is allocated. To send the code to the printer use CHR\$ function as follows -

```
PRINT #1, CHR$(n)
```

n is the character code and sends only the code which has meaning for the printer. The printer codes for when you enlarge the characters or compress them, etc., are all set. So, study the printer manual first and experiment yourself.

* Printer Page Feed

For printers we normally use stock form which has perforations.

The printer understands the position of the head as the head of the paper when power is switched on. There is no special command to specify the head of the paper.

By the way, a CLS execution clears the screen in a monitor, with the printer we cannot make the paper blank once it has been printed on. Instead of erasing what has been printed, we feed the paper to a new page. What command controls paper feed?

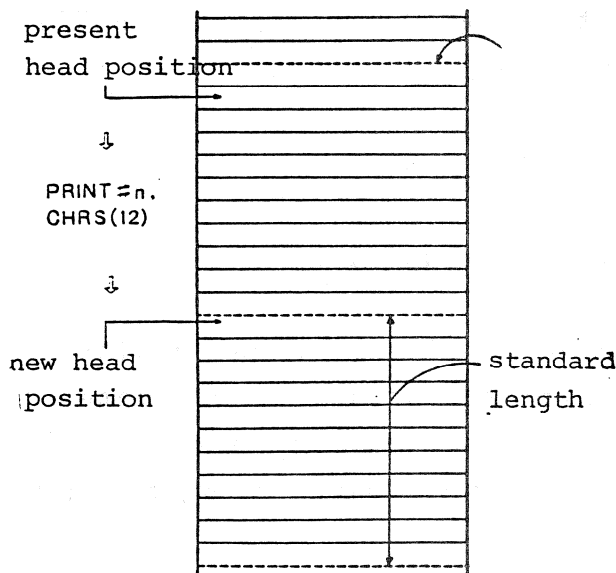


Fig. 13.2 Printer Paper and Page Feed

To clear the screen, we use CLS but PRINT CHR\$(12) has the same function since CHR\$(12) generates the screen clear code. With the printer, we do not have CLS so we use PRINT#n, CHR\$(12).

When the printer is turned on, if the perforation is set at the head of the paper, the paper is fed to the head of the next page no matter at which line you clear.

Let's now try a program using this function. In this program, we will feed to the new page first then display the title and output some numerals.

```

10 REM
20 REM          PRINTER SAMPLE PROGRAM
30 REM
50 OPEN"Q",#1,"LPT0:"
60 PRINT#1,CHR$(12)
70 PRINT#1,TAB(15);"** Printer Sample Program ***"
80 GOSUB 240
90 PRINT#1,TAB(7);"A";TAB(19);"B";
100 PRINT#1,TAB(28);"C=A*B";TAB(40);"C/3.14";
110 PRINT#1,TAB(54);"A+B";TAB(64);"C*3.14"
120 GOSUB 240
130 DIM A(10),B(10),C(10)
140 FOR I=1 TO 10
150 A=RND(1)*10:B=RND(1)*10
160 A(I)=A:B(I)=B:C(I)=A*B
170 NEXT
180 FOR I=1 TO 10
190 PRINT#1,USING"####.#####";A(I),B(I),C(I),C(I)/3.14,A(I)+B(I),C(I)*3.14
200 NEXT
210 GOSUB 240
220 CLOSE
230 END
240 FOR I=1 TO 79
250 PRINT#1,"-";
260 NEXT
270 PRINT#1
280 RETURN

```

To use the printer we really only need to remember these commands -

OPEN, CLOSE, PRINT#n, PRINT#n, USING

This is the result of our program -

```

          ** Printer Sample Program ***
-----
      A          B          C=A*B          C/3.14          A+B          C*3.14
-----
5.9106500  2.0799100  12.2936000  3.9151700  7.9905700  38.6020000
5.5096700  6.3586300  35.0340000  11.1573000  11.8683000  110.0070000
1.0411000  8.0633400   8.3947800   2.6735000   9.1044400   26.3596000
0.4290730  9.5386200   4.0927700   1.3034300   9.9676900   12.8513000
3.5428900  5.5649800  19.7161000   6.2790100   9.1078600   61.9085000
4.4577800  0.1312870   0.5852490   0.1863850   4.5890700    1.8376800
5.4344300  5.3166400  28.8929000   9.2015600  10.7511000   90.7237000
3.1640300  4.6643500  14.7582000   4.7000500   7.8283800   46.3406000
1.0290900  9.7981800  10.0832000   3.2112100  10.8273000   31.6613000
1.4422400  2.8198000   4.0668400   1.2951700   4.2620500   12.7699000
-----

```


* Outputting the Program List

We will explain here another important usage of the printer - the program list.

When we use the printer in a program we have to use OPEN and CLOSE to open and close the file. But, when you output a program list, we do not need such procedures. Simply write -

```
LIST "LPT0:"
```

To display a list on the screen, we only need to use a LIST command but with the printer we add the file descriptor LPT0:. Of course, you can designate the line number and range of the list -

```
LIST "LPT0:", 100-300
LIST "LPT0:", -350
LIST "LPT0:", 200
LIST "LPT0:", 100, 300
LIST "LPT0:", 300
```

CHAPTER FOURTEEN

I/O SLOTS

In Level-3, we can insert interface cards in the I/O slots and easily expand the functions. We will explain the I/O slot functions a little more fully here.

The Level-3 has six I/O slots. In these slots there are 56 terminals which provide the necessary signals between the MPU and interface. These terminals consist of address signal lines, data signal lines, control signal lines and power. The slot pin arrangement is shown below -

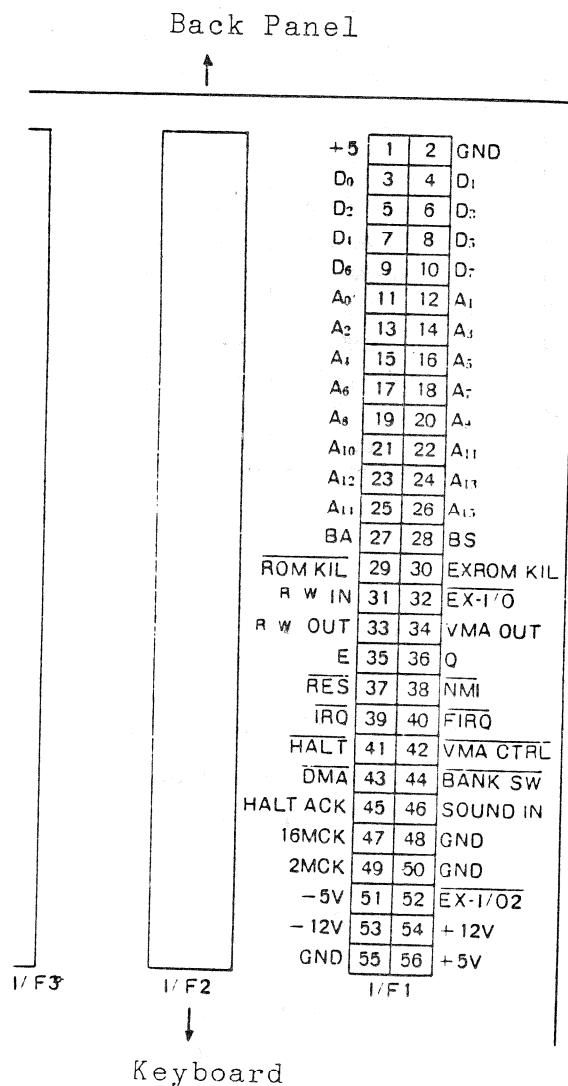


Fig. 14.1 I/O Slot Pin Arrangement

Table 14.2 Slot Pin Arrangement

direction cpu ←	sig.	pin no.	pin no.	sig.	direction cpu ←
→	+5V	1	2	GND	→
→	D ₀	3	4	D ₁	→
→	D ₂	5	6	D ₃	→
→	D ₄	7	8	D ₅	→
→	D ₆	9	10	D ₇	→
→	A ₀	11	12	A ₁	→
→	A ₂	13	14	A ₃	→
→	A ₄	15	16	A ₅	→
→	A ₆	17	18	A ₇	→
→	A ₈	19	20	A ₉	→
→	A ₁₀	21	22	A ₁₁	→
→	A ₁₂	23	24	A ₁₃	→
→	A ₁₄	25	26	A ₁₅	→
→	BA	27	28	BS	→
→	ROM KIL	29	30	EXROM KIL	→
→	R/W IN	31	32	EX-I/O	→
→	R/W OUT	33	34	VMA OUT	→
→	E	35	36	Q	→
→	RES	37	38	NMI	→
→	IRQ	39	40	FIRO	→
→	HALT	41	42	VMA CTRL	→
→	DMA	43	44	BANK SW	→
→	HALT ACK	45	46	SOUND IN	→
→	16MCK	47	48	GND	→
→	2MCK	49	50	GND	→
→	-5V	51	52	EX-I/O2	→
→	-12V	53	54	+12	→
→	GND	55	56	+5V	→

* Explanation of Each Terminal

The function of each terminal will be outlined in this section. The use of some of the terminals is not known but no doubt this will be disclosed in due course.

(1) +5V, -5V, +12V, -12V Power Terminals

To supply the necessary power to the interface. Pin 1, 56 are +5V, pin 51 is -5V, pin 54 is +12V and pin 53 is -12V. The power unit is the switching regulator type and four types

of power may be supplied depending on the need. The capacity of this power supply is not so big so that it is not possible to supply for outside use.

(2) $D_0 - D_7$ (Data Bus)

Data bus 8-bit and address bus 16-bit signals are connected. These signals are bi-directional thus the address can be given to the interface from the MPU or the memory can be accessed from the interface card.

(3) BA (Bus Available), BS (Bus Status)

As the MPU address bus, data bus and R/W signal are each at the floating level, the BA signal shows some outside device apart from the MPU, can occupy the bus. BA and BS are a pair and inform the MPU's conditions of use to outside devices.

- Interrupt acknowledge condition indicates the period the MPU reads the routine address when the interrupt vector address is output according to an interrupt request from outside.
- Sink acknowledge state is the period when the MPU receives an outside signal from the interrupt signal line after the SYNC command has been executed.
- Halt/bus ground state means the MPU is in the halt condition and the memory may be accessed freely from outside.

(4) ROM KIL, EXROM KIL

ROM KIL is connected to the ROM chip select in Level-3. When this signal is low, the ROM does not function and the ROM memory is released.

EXROM KIL is connected with the ROM on the interface card. With this signal, the ROM in the expansion memory card does

not function and avoids using the same bus.

(5) R/W IN, R/W OUT

The R/W signal shows the direction of movement of data on the data bus. When this signal is low, data moves from MPU to memory and when high, data is sent from memory to MPU. R/W OUT is used to access the memory on the interface card. The R/W signal in MPU is connected with R/W OUT.

R/W IN is used to access the memory in Level-3 from the interface card. In this case, set the MPU halt state and avoid using the same bus.

(6) VMA OUT, VMA CTRL

VMA OUT is the output of AND between BA and BS. With this signal we know the operating period of MPU and its stopped period, then we can avoid the access of outside devices and the MPU memory at the same time.

VMA CTRL controls the VMA OUT signal. When VMA CTRL is at low level, VMA OUT is always high at any condition, and if VMA CTRL is at high level, VMA OUT outputs as it is.

(7) E, Q

The E signal is the equivalent output signal of the 6800 ϕ_2 clock (or HD46800). The MPU reads the data at E's falling edge of the pulse.

The Q signal is a $\frac{1}{4}$ clock faster than E. The address Q output from MPU is decided at Q's rising edge of the pulse.

rising edge falling edge

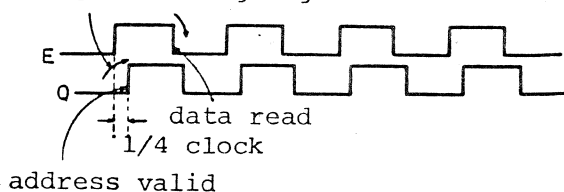


Fig. 14.3 E Signal and Q Signal

of power may be supplied depending on the need. The capacity of this power supply is not so big so that it is not possible to supply for outside use.

(2) $D_0 - D_7$ (Data Bus)

Data bus 8-bit and address bus 16-bit signals are connected. These signals are bi-directional thus the address can be given to the interface from the MPU or the memory can be accessed from the interface card.

(3) BA (Bus Available), BS (Bus Status)

As the MPU address bus, data bus and R/W signal are each at the floating level, the BA signal shows some outside device apart from the MPU, can occupy the bus. BA and BS are a pair and inform the MPU's conditions of use to outside devices.

- Interrupt acknowledge condition indicates the period the MPU reads the routine address when the interrupt vector address is output according to an interrupt request from outside.
- Sink acknowledge state is the period when the MPU receives an outside signal from the interrupt signal line after the SYNC command has been executed.
- Halt/bus ground state means the MPU is in the halt condition and the memory may be accessed freely from outside.

(4) $\overline{\text{ROM KIL}}$, EXROM KIL

$\overline{\text{ROM KIL}}$ is connected to the ROM chip select in Level-3. When this signal is low, the ROM does not function and the ROM memory is released.

EXFOM KIL is connected with the ROM on the interface card. With this signal, the ROM in the expansion memory card does

devices. The HALT ACK signal shows the MPU is in the halt state.

(13) $\overline{\text{DMA}}$

The $\overline{\text{DMA}}$ terminal sends a signal to stop the MPU executing command temporarily and releases the bus for other use. If this signal is low, the command execution is stopped at the completion of the machine cycle.

(14) SOUND IN

This terminal is connected to the speaker inside.

(15) 16MCK, 2MCK

A 16.128 MHz clock is output from 16 MCK and a 2.016 MHz clock is output from 2 MCK.

(16) $\overline{\text{BANK SW}}$, $\overline{\text{EX-I/O}}$, $\overline{\text{EX-I/O2}}$

We do not know the purpose of these signals at the moment.

(17) GND

This is the ground line, or common earth line, to each terminal.

CHAPTER 15

THE RS-232C INTERFACE

The RS-232C interface is standard equipment in the Level-3 to use in connection with outside devices and the data transfer between equipment. This interface has a standard specification and so we can use it with other equipment that has the same interface by connecting it with just one cable. The functions and method of use of the RS-232C interface follow.

1. RS-232C Interface Data Transfer

The RS-232C type of data transfer is in serial form like a cassette tape. One block consists of eight bits and one bit at a time is sent in order.

The actual data comprises an 8-bit data bit, a start bit and stop bit, as shown in Fig. 15.1.

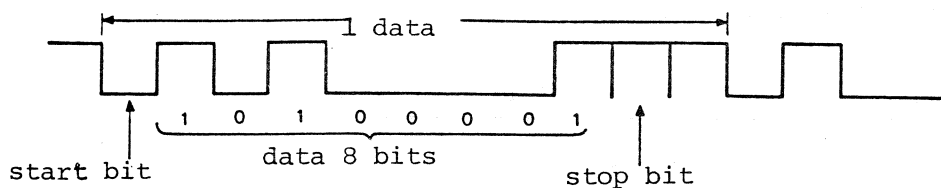


Fig. 15.1 Bit Construction of One Block of Data

The start bit is, as indicated by its name, the signal to start the data and this is achieved by changing the high level signal to low level.

Data is sent after this and the next start bit preparation is carried out at the last part in the stop bit. The number of stop bits is either one or two depending on the device.

Level-3 can select the length of the stop bit with software. But the length of one bit is decided by the hardware, and this is decided by the data transfer speed.

The transfer speed is decided by the clock to the ACIA (Asynchronous Communication Interface Adapter). The transfer speed is called the baud rate (the number of bits to be transferred in one second). The Level-3 supports from 300 bauds to 4,800 bauds which can be selected at will.

The chip switch (CS3) sets the baud rate factor. When you remove the upper cover, you should be able to see the chip switches to the left of the memory expansion connector.

The left switch sets the baud rate factor. The numbers 1-4 are printed on the board. The relationship between these numbers and baud rate is as per Table 15.2. The switch only gives the basic frequency to ACIA. Slow and fast are software switchable. For example, if you want to set to 4,800 baud, set the chip switch to 3 and select Fast mode by software. Setting the chip switch is a little bit difficult as the connector gets in the way so it is better to use tweezers or small radio pinchers.

Table 15.2 Chip Switch Numbers and Baud Rate

Soft Hard	Slow	Fast
1	300 baud	1200 baud
2	600 baud	2400 baud
3	1200 baud	4800 baud
4	2400 baud	-----

2. RS-232C Interface Usage

There are two ways to use the RS-232C interface. The first way is to use it as a terminal. You have seen from photographs and such that the CRT terminal and console typewriter are connected with the large main frame.

Level-3 can do the same job. We call this usage terminal mode. In terminal mode, we do not use Level-3 as a computer but utilise only the keyboard and screen.

The second method is the mode where it is considered an outside file in the program and communication between the Level-3 and the outside device is conducted. When this mode is used, we can carry out decentralised processing (satellite processing).

* Using Level-3 as a Terminal (TERM command)

Terminal mode uses the Level-3 as an I/O device. Naturally we have to ready another computer for such an application. One we could connect with is the S100 bus microcomputer which has Z-80, 64K byte memory, double-sided 8" floppy and CP/M.

Another is the PDP-11/34 with 256 Kbyte memory, two 5.5 M byte hard disk and RSX-11/M OS.

To connect these two computers, all we need is one RS-232C cable.

* Connecting the Level-3 to a Microcomputer

Let's connect the Level-3 to the microcomputer. Connect the cable to the 25-pin connector on the back panel. See Fig. 15.3 below.

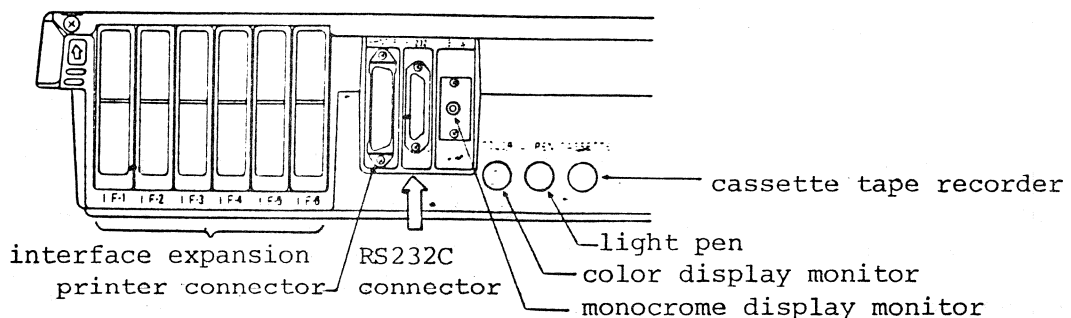


Fig. 15.3 Level-3 Back Panel

Then we prepare the software. Level-3 has a TERM (TERMinal command). By adding an option to TERM, we have to make it communicate efficiently with other systems.

The first problem is setting the baud rate. Our speed is 4,800 baud so we set the chip switch in level-3 to 3. Table 15.2 shows 4,800 baud can be obtained by the Fast mode for Switch 3.

TERM "F8N1F"

The actual designation if effected with the above example. The character string enclosed by " is the option designation. This first letter of the designation is called the clock designation and switches either to fast or slow mode. We have set it to F (fast mode) as we are using 4,800 baud. The second 8 is bit designation which indicates the length of the transfer word. We must match this with the length of one data which the other apparatus requests. In this case, the other computer requests 8 bit and consequently we set 8. Sometimes the equipment requests 7 bit in which case Level-3 can select either 8 or 7.

The third N is the parity bit designation. We can set N, E or O. Parity checks whether the transferred data has been correctly received by the other machine. (Note that data must be 7 bits if parity is required.)

First we check how many number 1s are in a piece of data received. Depending on the content of the data, the number is judged to be odd or even.

By making the parity bit H(1) or L(0), we can always make the H(1) number an odd or even number. For example, we set an odd number for the transfer logic between the pieces of equipment. We set the parity bit to make the H(1) number odd and send this data. If the receiver acknowledges its number H(1) as odd, we will know that the data has been correctly sent. We call this odd parity.

We can also have even parity if the data has an even number system. On some occasions, parity is not required and we distinguish between three states -

O (ODD), E (EVEN), N (NONE)

The fourth part of the TERM designation is for the stop bit. As each piece of equipment has a different stop bit length you need to check whether the stop bit is 1 or 2 bits.

The last is called mode designation where we select either full-duplex or half-duplex.

Full-duplex mode allows us to transceive independently. That is, when you only press a key, that character does not appear on the screen. If there is an output request from the other machine, however, it is output on the screen (echo back). With the half-duplex, only input or output can be done at the one time. There is no echo back so that the sending machine can output its input characters on the screen by itself.

From the point of the host computer, if I/O can be independently conducted, this increases its flexibility which can only mean good. Most other equipment adopts the fully-duplex method.

We designate F for fully-duplex and H for half-duplex.

Keeping the above in mind, look at the TERM command once more.

TERM "F 8 N 1 F"
 ↑
 clock
 ↑
 data bit length
 ↑
 parity
 ↑
 stop bit
 ↑
 mode

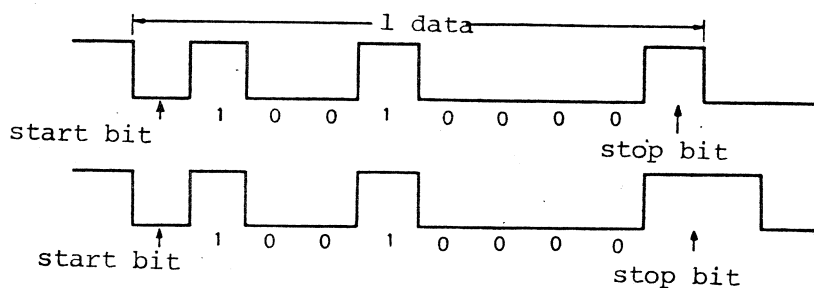


Fig. 15.4 Data and Stop Bit

From this we understand the conditions set for the terminal are fast mode, 8-bit data, non parity, stop bit is 1 and the mode is the fully-duplex method.

Not every condition for setting needs to be included as per Table 15.5. If omitted, Level-3 presumes that you have designated Level-3's internal setting conditions.

Table 15.5 Conditions at Time of Delivery from Factory

Condition	Set Character	If Omitted
Clock	S or F	S
Bit Length	7 or 8	Set by Dip Switch No. 7 Normally 8
Parity	N, E or O	N
Stop Bit	1 or 2	2
Mode	F or H	Set by Dip Switch No. 6 Normally F

Bit length and mode can be reset with the internal dip switch to the omitted condition but when shipped from the factory they are set as shown in this table.

* Connecting Level-3 to a Minicomputer

We will try connecting Level-3 to PDP-11/34 next. We set the baud rate to a relatively slow 2,400 which means we set the chip switch to 4 and enable slow mode. If the chip switch is set to 2 and used in fast mode, we would get the same result so it does not matter which you choose.

TERM "S8N1F"

The PDP-11/34 starts data transfer under the conditions set here.

As Level-3 has the function keys the most commonly used commands would be set with these keys and put on line with the TERM command. The cursor key cannot be used but the repeat function and CTRL key function remain.

* When Level-3 does not work as a Terminal

Even if you connect the cable and set the baud rate, you may not be able to use Level-3 as a terminal. This is because the logic does not match. The chip switch (CS4) in the right of the baud rate chip switch (CS3) sets the transceive logic. The RS-232C interface has six signal lines to transceive data and these are outlined for you below -

(1) TXD (Transmitted data) - output

This signal sends the data to outside devices. Set the chip switch to 8 or 7. If set to 8, it becomes positive logic and if 7, negative logic. With negative logic, if 1 is output, the connector has 0 as output and if 0 is output, 1 is output to the connector. That is to say, the reverse logic appears. Normally, the chip switch is 7 or negative logic.

(2) RXD (Received data) - input

Data input line from outside. This is not in the chip switch and the logic cannot be set.

(3) RTS (Request to send) - output

This is the data request line to outside equipment. This line informs the other that preparation has been completed. This signal is output from Level-3. The chip switch is 5 or 6. Normally this signal is used in negative logic - 5 being positive logic and 6 being negative logic.

(4) DCD (Data carrier detect) - input

This signal line shows that there is transmission from outside. Level-3 starts receiving data with this signal. The chip switch is 1 or 2. 1 is the earth state and 2 is active condition.

(5) CTS (Clear to send) - input

This signal acknowledges that outside equipment can receive data at any time. With this signal Level-3 sends the next data. The chip switch is 4 or 3. 4 is the active state and 3, the earth state.

(6) GND (Ground) - earth

The sixth signal is earth. This line gives each signal the potential difference.

The RS-232C has five lines (6 including earth). We need to check what logic the other equipment requires.

Mostly, everything should work but if it does not, check the input/output pin arrangement. The signal pins in the Level-3 connector are set out in Table 15.6 below.

Table 15.6 Level-3 Connector Pins

Signal	Pin No.	Send/Receive
T X D	2	Sending
R X D	3	Receiving
R T S	4	Sending
C T S	5	Receiving
D C D	8	Receiving
G N D	7	-

The Level-3 RS-232C interface, when there is input, stores the input data in the communication buffer in the memory and takes out one character and displays it on the screen.

If the Level-3 screen is in high resolution mode, the scrolling time takes time, thus this buffer becomes full quite quickly. The Level-3 displays a 'Buffer Overflow' error message and returns to BASIC mode. When this happens, set the screen to normal mode (SCREEN 0) and execute TERM again.

When TERM is executed and a 'Device I/O Error' message appears on the screen this means the data has not been received correctly. In this case, execute the following program -

```
10 ON ERROR GOTO 30
```

```
20 TERM"_____"
```

```
30 RESUME
```

↑
Condition designation

If the error occurs at TERM, line 30 is executed and we return to the line where the error occurred (that is, line number 20 TERM). With this, we can prevent the computer being stopped by an error.

When you CLOSE the dip switch No. 1 (set it to 0), we get terminal mode as soon as the power comes on. However an error may occur from the other side with the result that the data is not being transferred properly at first. Therefore, it is wise to insert this program. It is not so long so it should not be too troublesome to input it by hand each time.

3. RS-232C Interface Program Operation

The RS-232C interface is used not only in terminal mode but also may be operated by software, so that its application becomes considerably wider.

* OPEN, CLOSE

Firstly, to open the RS-232C communication file, we use OPEN. The file descriptor is COM 0:.. The operation modes are either I (input) or O (output). The option is the character string condition setting mentioned in the TERM section.

```
OPEN "I", (#)n, "COM 0 : ( --- )"
      or      ↑           ↑
      O      file number conditions.set
      ↑
operation mode
n is 1-16
```



```
OPEN "I", 1, "COM 0 : (F8N1)"
OPEN "O", 2, "COM 0 : (S7E2)"
```

Note that OPEN does not designate fully-duplex or half-duplex. After the execution has been completed, close the open file with CLOSE.

```
CLOSE #(n, n1, .....)
```

n is the file number used in OPEN. If you execute only CLOSE, all the files are closed. When n is designated, only that particular file is closed. We can write many n's separating them with (,). # can be omitted.

```
CLOSE
CLOSE #1
CLOSE 1, 2, 3, 4
```

Now the data transfer is ready. Let's actually input/output the data.

```
* INPUT#, PRINT#
```

Input from the RS-232C circuit is INPUT#; output is PRINT#. However, to input, we must OPEN with I mode and to output, we also must OPEN with O mode.

The file number set by the OPEN statement is added on after #. OPEN is one statement which cannot designate I and O at the same time. When you carry out input/output in the program, you need two OPENs. INPUT# and PRINT# are equivalent to the INPUT statement and PRINT statement functionally and there is no change in writing the variables, and (,) or (;).

```
10 OPEN "I", #1, "COM0:(F8N1)"
20 INPUT #1, A$
```

```

10 OPEN"O",#2,"COMO:(F8N1)"
20 PRINT#2,D$
  $

```

```

10 OPEN"O",#1,"COMO:(F8N1)"
20 OPEN"I",#2,"COMO:(F8N1)"
30 INPUT#2,A$
  :
100 PRINT#1,K$,MID$(X$,I,2)
  :
300 CLOSE#1,#2

```

Of course, after OPEN, you can use all the statements and functions in BASIC.

We can also include the file descriptor in the character variables so that special usage is possible.

```

10 A$="COMO:(F8N1)" .... Set file descriptor in A$
20 OPEN"O",#1,A$      .... OPEN RS-232C
30 PRINT#1,X$
  :
  :
500 CLOSE#1

```

In this program, you can change COMO: in line 10 to LPTO: or CASO:. Input/output commands such as INPUT#, PRINT#, LINE INPUT#n, PRINT#n, USING may also be used, but there is no INPUT WAIT for RS-232C.

We can also use INPUT\$(n, #n) as an input function.

* RS-232C Interrupt Control

RS-232C has interrupt execution like the keys and light pen. We should execute the following command as preparation -

ON COM(0) GOSUB n n is the procedure routine line number



First of all, we declare what we should do if interrupt occurs. The *n* in GOSUB *n* is the subroutine head line number for when the interrupt occurs, that is when data is sent.

As soon as the interrupt occurs, the program execution is stopped and we jump to line number *n*.

Next, to make the RS-232C interface interrupt possible, execute -

```
COM(0) ON
```

ON stands for permission. If we have OFF, it means prohibited and STOP means stop. STOP and OFF appear to be the same but with STOP the computer does not jump to the subroutine if the input has taken place but, for the time being, accepts the input and remembers that the interrupt has occurred. Therefore, if we execute COM(0) ON after this, the designated program is executed.

The preparation for interrupt is ready now -

```
10 ON COM(0) GOSUB 100
20 COM(0) ON
30
    {
100
    {
200 RETURN
```

} interrupt procedure program

Return to the main routine from the interrupt is accomplished by RETURN. If we have had interrupt to read that input data, we must, of course, open the file. That is, ON COM GOSUB only judges whether there is input or not.

```
10 ON COM(0) GOSUB 100
20 COM(0) ON
30 OPEN "I", #1, "COM0:(F8N1)"
40 GOTO 40
```

..... dummy program

```
100 PRINT"Interrupt:RS-232C"  
110 INPUT#1,A$  
120 PRINT"Receive data is ";A$  
130 PRINT"now return"  
140 RETURN
```

The RS-232C in program mode essentially uses the above commands. Interrupt can be used and satellite systems can be organised if you connect up with computer systems. However, as Level-3 functions in BASIC, if you have a complicated job the execution time becomes longer and you may not be able to communicate with other computers satisfactorily.

In this kind of case, you really need to establish some waiting time with the host computer to make sure that data transfer works smoothly.

* RS-232C Connection to Printer and other Measuring Equipment

The RS-232C can also work with printers, X-Y plotters and other devices. We can connect Level-3 quite easily and effect control equally easily. Particularly the RS-232C printer can be used in the same way as a normal printer. By changing LPT0: to COM0:, LIST output is possible too.

```
LIST "COM 0 : ( )"  
LIST "COM 0 : ( )", 300-1000  
LIST "COM 0 : ( )", 500-
```

Handling the RS-232C interface is not at all difficult so if you have the opportunity to use it, we recommend you try!

SUPPLEMENTTable S.1 Decimals. Binarys. Hexadecimals.Octals Correspondence

Decimal	Binary	Hexadecimal	Octal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17
16	10000	10	20

SUPPLEMENT

Table S.2 I/O Map (&HFF00 - &HFFEF)

Address	Corresponding Hardware
FF00 - FFBF	Open
FFC0 - FFC3	Parallel interface
FFC4 - FFC5	Serial interface
FFC6 - FFC7	CRT controller
FFC8	Keyboard interrupt ($\overline{\text{NMI}}$)
FFC9	Internal dip switch
FFCA	Timer interrupt ($\overline{\text{IRQ}}$)
FFCC = FFCE	Open
FFD0	Mode select
FFD2	Remote control
FFD3	Speaker control
FFD4	Timer mask
FFD5	Light pen register
FFD6	Interlace control
FFD7	Open
FFD8	Colour register
FFD9 - FFDF	Open
FFE0	Keyboard
FFEI - FFEF	Open

Table S.3 BASIC ROM Subroutine (only disclosed ones)

FAB3	Cold start
A438	} Warm start
A3A7	
E804	Input one character
E820	Output one character

SUPPLEMENT

Table S.4 Level-3 BASIC Work Area

Address	Content
0006	Floating accumulator type
001D-001E	BASIC program start address
001F-0020.	Simple variable start address
0021-0022	Array start address
0023-0024	Free area start address
0025-0026	Stack start
0027-0028	Output work
0029-002A	Output work
002B-002C	BASIC program end address
002D-002E	BASIC PROGRAM COUNTER
0056-005E	Floating point accumulator
009E	File number
00A2	Maximum horizontal character number
00A3-00A4	Display area start address
00A5-00A6	Display area end address
00A7	Graphic mode
0115	CURSOR MODE
0116	Character colour
0148-0149	EXEC execution address
014A-015D	USR jump table
01C8-01C9	Work area start
01CA-01CB	Key buffer start
01CC-01CD	Function key table start
01CE-01CF	Function key table end
01D0-01D1	Cassette buffer start
01D2-01D3	Input/output device table start
01F4	Remote state
01FB	Item
01FC	Type
01FD	File name exist or not
01FE-0205	File name
0206	Input/output device number
216-21E	Timer
21F0	Character type

716-Acc- Timer ($\frac{1}{60}$ second)

71C-221 date

24F-250 - ML Monitor counter

BS-Input mode
B6

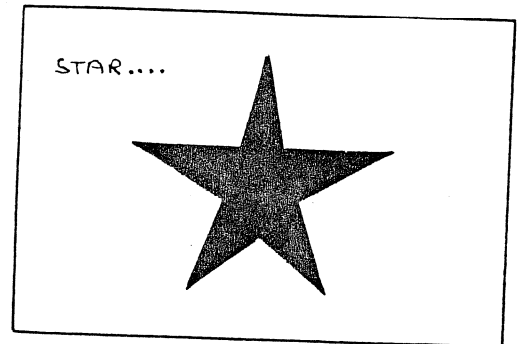
SUPPLEMENTSSAMPLE PROGRAMS

PROGRAM 1;

```

100 SCREEN 1,,0:COLOR 4,0:CLS
110 INPUT "SCREEN WIDTH",SW
120 WIDTH SW
130 READ XO,YO
140 LINE@ (XO,YO)-(XO,YO),PSET,6
150 FOR I=1 TO 9
160 READ X,Y
170 LINE@-(X,Y),PSET,6
180 NEXT I
190 LINE@-(XO,YO),PSET,6
200 PAINT@ (320,100),6,6
210 LOCATE 0,0:PRINT " STAR .... "
220 GOTO 220
230 END
240 DATA 320,20,290,70,150,70,270,100,23
0,150,320,120,410,150,370,100,490,70,350
,70

```



PROGRAM 2: BASIC VERSION TEST

If you have sum check &H08 and Parity &HF4,
you have the same version machine as I have!

```

100 'Level-3 BASIC version test
110 FORA=&HA000 TO&HFEFF
120 U=A/256
130 IF (U-INT(U))=0 THEN PRINT HEX$(A)+" ";
140 GOSUB 230
150 NEXTA
160 FORA=&HFFFO TO&HFFFF
170 GOSUB 230
180 NEXTA
190 BEEP
200 PRINT "ROM sumcheck &H";RIGHT$("0"+HEX$(SZ),2)
210 PRINT "    parity    &H";RIGHT$("0"+HEX$(VZ),2)
220 END
230 I%=PEEK(A)
240 SZ=(SZ+I%)/MOD256
250 VZ=VZXOR I%
260 RETURN

```

Ready

RUN

```

A000 A100 A200 A300 A400 A500 A600 A700
AB00 A900 AA00 AB00 AC00 AD00 AE00 AF00
B000 B100 B200 B300 B400 B500 B600 B700
B800 B900 BA00 BB00 BC00 BD00 BE00 BF00
C000 C100 C200 C300 C400 C500 C600 C700
CB00 C900 CA00 CB00 CC00 CD00 CE00 CF00
D000 D100 D200 D300 D400 D500 D600 D700
D800 D900 DA00 DB00 DC00 DD00 DE00 DF00
E000 E100 E200 E300 E400 E500 E600 E700
EB00 E900 EA00 EB00 EC00 ED00 EE00 EF00
F000 F100 F200 F300 F400 F500 F600 F700
FB00 F900 FA00 FB00 FC00 FD00 FE00
ROM sumcheck &H08
    parity    &HF4

```


PROGRAM3; Level-3 character set;

```

100 SCREEN 0,,0:WIDTH 40
110 LOCATE 2,0
120 FOR UP=0 TO 15
130 PRINT HEX$(UP); " ";
140 NEXT UP
150 PRINT:PRINT
160 FOR LOW=0 TO 15
170 PRINT HEX$(LOW);
180 FOR UP=0 TO 15
190 CODE=UP*16+LOW
200 IF CODE<32 THEN C$="."
    ELSE C$=CHR$(CODE)
210 PRINT " ";C$;
220 NEXT UP.
230 PRINT
240 NEXT LOW

```

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	.	.	.	0	2	P	.	p	x	x	+	-	ラ	ア	カ	キ
1	.	.	.	1	A	Q	a	q	月	火	.	ア	チ	シ	ホ	フ
2	.	.	.	2	B	R	b	r	日	水	.	イ	ツ	メ	リ	ル
3	.	.	.	3	C	S	c	s	木	木	.	ウ	フ	セ	セ	セ
4	.	.	.	4	D	T	d	t	金	金	.	エ	ト	ト	ト	ト
5	.	.	.	5	E	U	e	u	土	土	.	オ	ナ	ナ	ナ	ナ
6	.	.	.	6	F	V	f	v	土	土	.	カ	ナ	ナ	ナ	ナ
7	.	.	.	7	G	W	w	w	土	土	.	ア	ナ	ナ	ナ	ナ
8	.	.	.	8	H	X	h	x	土	土	.	イ	ツ	メ	リ	ル
9	.	.	.	9	I	Y	i	y	土	土	.	ウ	フ	セ	セ	セ
A	.	.	.	A	J	Z	j	z	土	土	.	エ	コ	ハ	ハ	ハ
B	.	.	.	B	土	土	.	ア	ナ	ナ	ナ	ナ
C	.	.	.	C	土	土	.	イ	ツ	メ	リ	ル
D	.	.	.	D	土	土	.	ウ	フ	セ	セ	セ
E	.	.	.	E	土	土	.	エ	コ	ハ	ハ	ハ
F	.	.	.	F	土	土	.	ア	ナ	ナ	ナ	ナ

Input SCREEN , ,1 and see the difference of non-interlace mode.

PROGRAM 4; color

This is the simple program. Hit **SHIFT** + **J** and see the difference.

```

110 FOR C=0 TO 15
120 COLOR C,0
130 FOR I=1 TO 4
140 PRINT "COLOR";C;
150 NEXT I
160 PRINT
170 NEXT C
180 COLOR 7,0

```

PROGRAM 5; Character function usage program.

```

100 CLEAR 4000:WIDTH 80:COLOR 5
110 DIM A$(1000)
120 FOR I=0 TO 1000
130 INPUT "WORD";A$(I)
140 IF A$(I)<>" " THEN NEXT I
150 TIME$="00:00:00"
160 PRINT:PRINT "Sorting.... Wait for a
moment please."
170 PRINT "Number of data" I-1
180 FOR J=I-1 TO 0 STEP -1
190 B$=""
200 FOR K=J TO 0 STEP -1
210 IF A$(K)>B$ THEN MAX=K:B$=A$(K)
220 NEXT K
230 SWAP A$(J),A$(MAX)
240 NEXT J
250 TM$=TIME$
260 TM=VAL(MID$(TM$,4))*60+VAL(RIGHT$(TM$,2))
270 FOR J=0 TO I-1
280 PRINT LEFT$(A$(J)+SPACE$(8),8);SPC(2);
290 NEXT J
300 PRINT:PRINT "Working time"RIGHT$(SPACE$(4)+STR$(TM),4)" seconds."
310 END

```

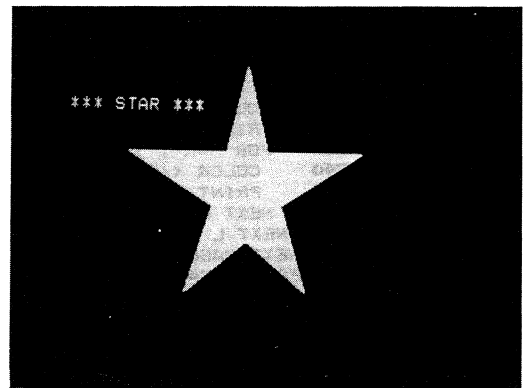
PROGRAM 7; Street car

PROGRAM 8: Super Graphics

```

100 '***** Graphic Mode *****
110 SCREEN 1,,1:COLOR 4,0:CLS
120 LOCATE 0,0:INPUT"MODE ";M
130 IF M=0 THEN SCREEN 1:WIDTH80:END
140 IF M<1 OR M>4 THEN 120
150 ON M GOSUB 310,320,330,340
160 '***** Plot Star *****
170 READ X0,Y0
180 LINE@ (X0,Y0)-(X0,Y0),PSET,6
190 FOR I=1 TO 9
200 READ X,Y
210 LINE@ (X,Y),PSET,6
220 NEXT I
230 LINE@ (X0,Y0),PSET,6
240 PAINT(320,100),6,6
250 RESTORE:GOTO 120
260 '***** Data of Star *****
270 DATA 320,20,290,70,150,70,270,100
280 DATA 230,150,320,120,410,150
290 DATA 370,100,490,70,350,70
300 '***** Mode change *****
310 SCREEN 0:WIDTH40:RETURN:" MODE 1
320 SCREEN 0:WIDTH80:RETURN:" MODE 2
330 SCREEN 1:WIDTH40:RETURN:" MODE 3
340 SCREEN 1:WIDTH80:RETURN:" MODE 4

```

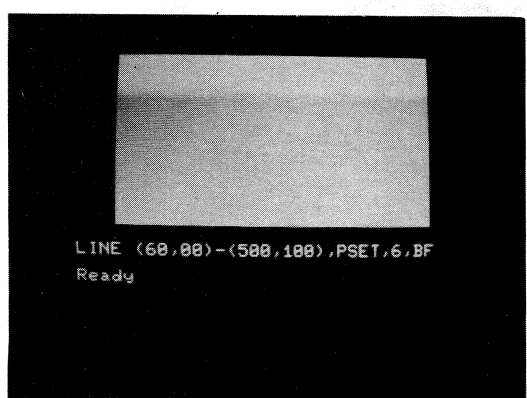
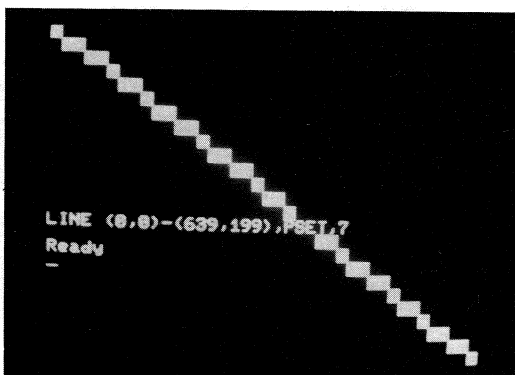
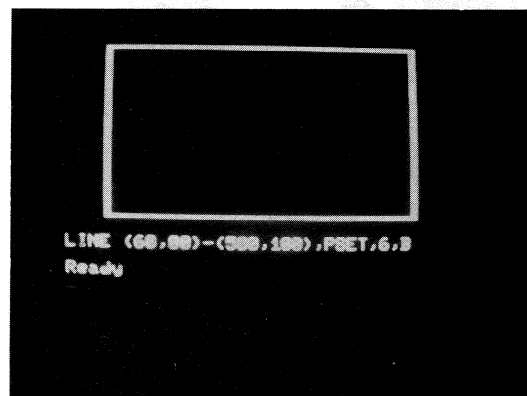
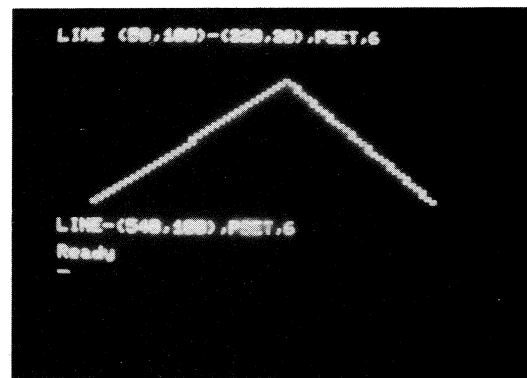
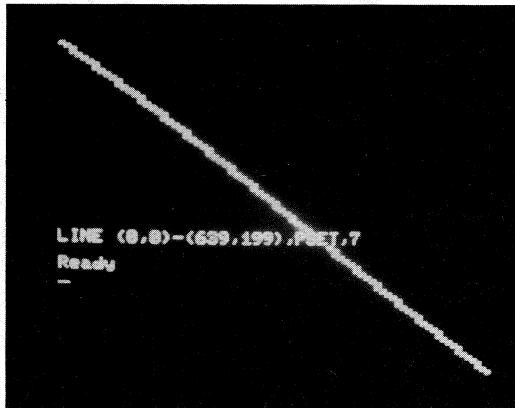


PROGRAM 9: PSET & PRESET statements;

Input following program and enjoy the color;

COLOR 7:SCREEN 0	→	CLS
PSET(320,100)		PSET(320,100)
COLOR 4,1		
	↙	
COLOR 7:SCREEN 0	→	PSET (320,102,10)
PSET(320,100,4)	→	
PRESET(320,100)	↙	PRESET(320,100,12)
	→	
PRESET(320,100,1)	↙	

PROGRAM 10; Usage of LINE function

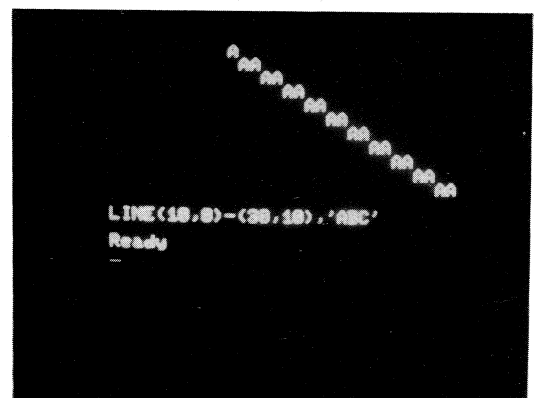
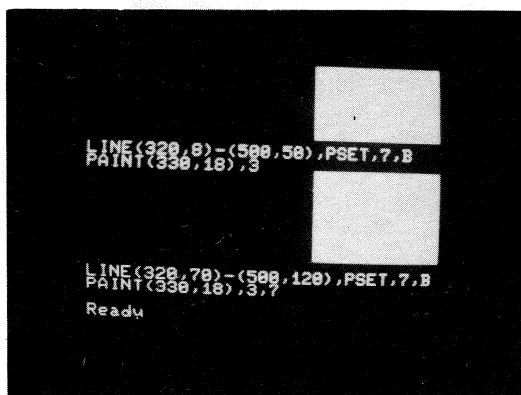
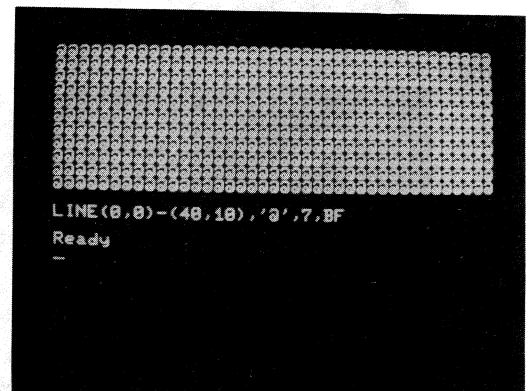
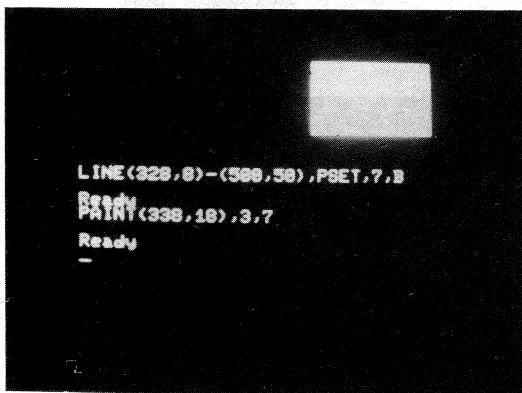
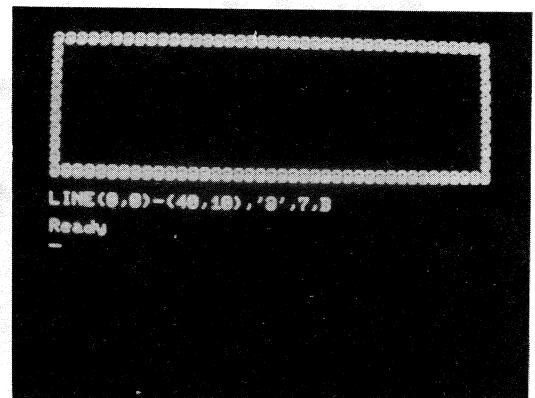
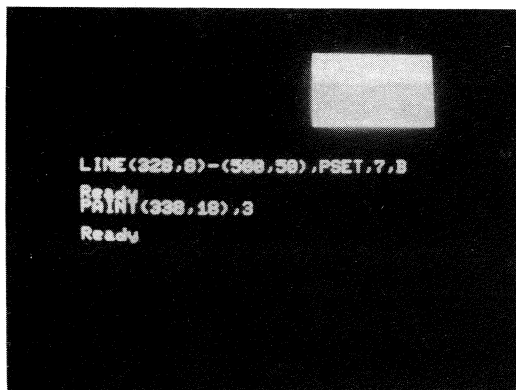
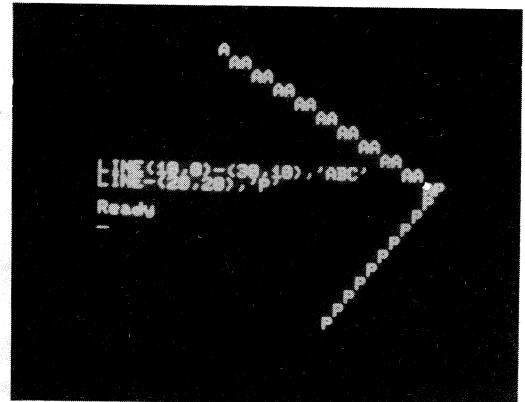


PROGRAM 11; LINE function program

```

Key in;
COLOR 7:SCREEN 0
PSET(320,100,1)
COLOR 4
LINE(0,0)-(639,199),PSET
LINE(639,0)-(0,199),PSET
LINE(0,0)-(639,199),PSET,10

```

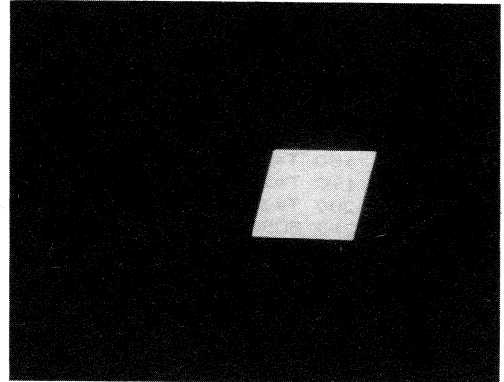


PROGRAM 12: Paint

```

100 '*** PAINT ***
110 SCREEN 1
120 LINE(320,0)-(639,0),PSET,1
130 LINE-(499,99),PSET,1
140 LINE-(180,99),PSET,1
150 LINE-(320,0),PSET,1
160 PAINT(635,1),1

```



PROGRAM 13: Sine Graphics

```

100 '*** SINE Graph ***
110 COLOR 7,0:SCREEN 1,,0:WIDTH80
120 DEF FNFC(X)=SIN(X)
130 PAI=3.141593

150 LINE(320,0)-(320,199),PSET
160 LINE(0,100)-(639,100),PSET

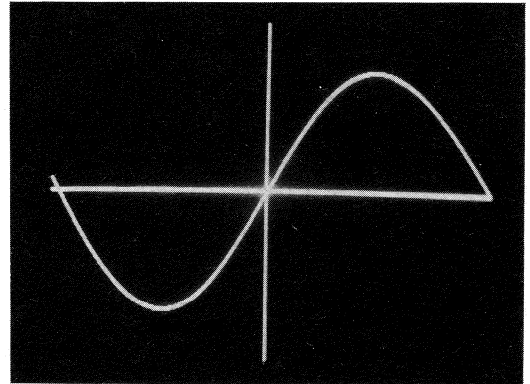
180 X=-PAI:Y=FNFC(X)
190 GOSUB 320
200 LINE(DX,DY)-(DX,DY),PSET,6

220 LOCATE 0,2:PRINT"y=sinx"
230 FOR X=-PAI TO PAI STEP PAI/90
240 Y=FNFC(X)
250 GOSUB 280
260 NEXT
270 END

290 GOSUB 320
300 LINE-(DX,DY),PSET,6
310 RETURN

330 DX=100*X+320
340 DY=-80*Y+100
350 RETURN

```



PROGRAM 14; Cosine & Tangent Graphics
Change Program 13 as follows;

```

120 DEF FNFC(X)=COS(X)

```

```

120 DEF FNFC(X)=TAN(X)
292 IF DY<0 OR DY>199 THEN P=1:RETURN
294 IF P THEN 312
312 P=0:LINE(DX,DY)-(DX,DY),PSET,6
314 RETURN
340 DY=-10*Y+100

```

```

120 DEF FNFC(X)=EXP(X)
160 LINE(0,150)-(639,150),PSET
340 DY=-10*Y+150

```

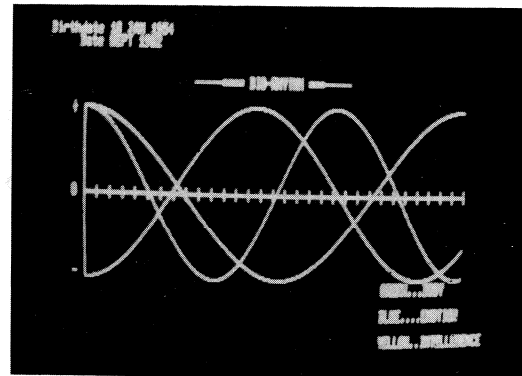
PROGRAM 14: Biorhythm

```

100 '*** BIORHYTHM ***
110 COLOR 7,0:SCREEN 1:WIDTH80
120 LINEINPUT"Birth Date ";D$
130 GOSUB 230:DD=DT
140 LINEINPUT"      Date ";D$
150 D$=D$+"/01"
160 GOSUB 230:DD=DT-DD
170 GOSUB 380

180 T=23:GOSUB 540:' BODY
190 T=28:GOSUB 540:' EMOTION
200 T=33:GOSUB 540:' INTELLIGENCE
210 GOSUB 660
220 LOCATE 0,2:END
230 '*** DAYS      ***
240 Y=VAL(D$)
250 D$=MID$(D$,INSTR(D$,"/")+1)
260 M=VAL(D$)
270 D$=MID$(D$,INSTR(D$,"/")+1)
280 D=VAL(D$)
290 YY=Y-1
300 DT=YY*365+YY*4-YY*100+YY*400
310 DT=DT+(M-1)*30+M*3.9*7+D
320 IF M>2 THEN DT=DT-2 ELSE 370
330 IF Y MOD 4 THEN 370
340 IF Y MOD 100 THEN 360
350 IF Y MOD 400 THEN 370
360 DT=DT+1
370 RETURN
380 '*** GRAPHICS 1 ***
390 LOCATE 31,4
400 PRINT"****BIORHYTHM****"
410 LOCATE 60,20:COLOR 4
420 PRINT"- : BODY"
430 LOCATE 60,22:COLOR 5
440 PRINT"- : EMOTION"
450 LOCATE 60,24:COLOR 6
460 PRINT"- : INTEL":COLOR 7
470 LINE(96,48)-(96,152),PSET,7
480 FOR I=0 TO 30
490 X=I*16+104
500 LINE(X,96)-(X,103),PSET,7
510 NEXT
520 PAI=3.141593
530 RETURN
540 ' SING WAVE
550 SP=(DD MOD T)/T*2*PAI
560 Y=-50*SIN(SP)+100
570 LINE(104,Y)-(104,Y),PSET,T*5
580 FOR D=0 TO 30
590 X=D/T*2*PAI
600 Y=SIN(X*SP)
610 'LOCATE 0,4:PRINTD,X,Y
620 LINE-(D*16+104,-50*Y+100),PSET,T*5
630 'D$=INPUT$(1)
640 NEXT
650 RETURN
660 '*** GRAPHICS 2 ***
670 LINE(88,100)-(591,100),PSET,7
680 FOR I=1 TO 10
690 READ X,Y,D$
700 LOCATE X,Y:PRINTD$
710 NEXT
720 RETURN
730 DATA 5.8,+,5.12,0,5.16,-,13,13,1
740 DATA 21,13,5.30,13,10,40,13,15
750 DATA 50,13,20,60,13,25,70,13,30

```



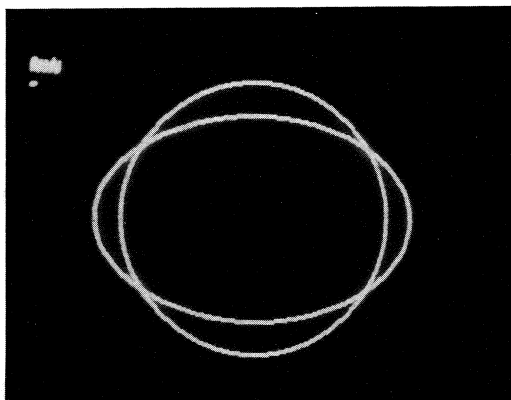
PROGRAM 15; Other graphics;

```

100 *** DRAW CIRCLE ***
110 SCREEN 1:WIDTH 80
120 PAI=3.141593
130 FOR TH=0 TO 2*PAI STEP PAI/90
150 X=2.4*80*COS(TH):Y=80*SIN(TH),
160 PSET(X+320,Y+100,3)

180 X=3.8*60*COS(TH):Y=60*SIN(TH)
190 PSET(X+320,Y+100,5)
200 NEXT

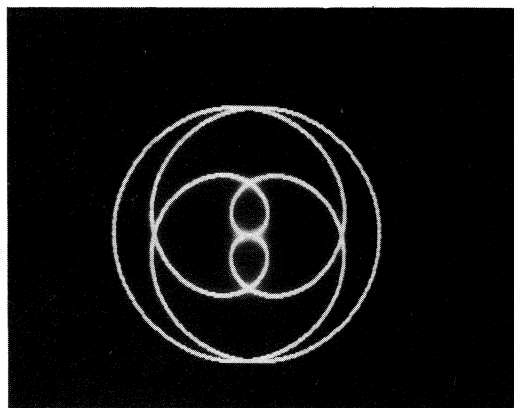
```



```

110 SCREEN 1:WIDTH 80
120 PAI=3.141593
130 FOR TH=0 TO 8*PAI STEP PAI/90
140 R=80*COS(TH/4)
150 X=2.4*R*COS(TH):Y=R*SIN(TH)
160 PSET(X+320,Y+100,2)
170 NEXT

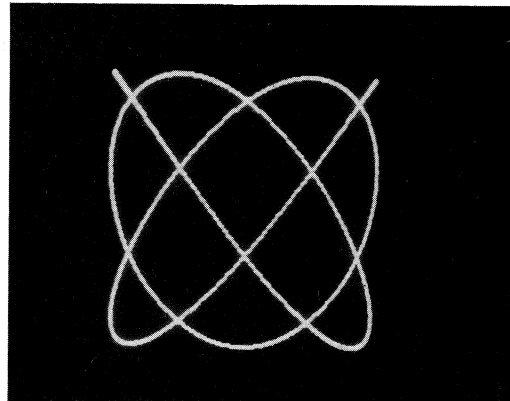
```



```

110 SCREEN 1:WIDTH 80
120 PAI=3.141593
130 A=5:B=6:V=A*B
140 FOR TH=0 TO V*PAI STEP PAI*A/90
150 X=2.4*80*SIN(TH/B-PAI/2)
160 Y=80*COS(TH/A)
170 PSET(X+320,-Y+100,4)
180 NEXT

```



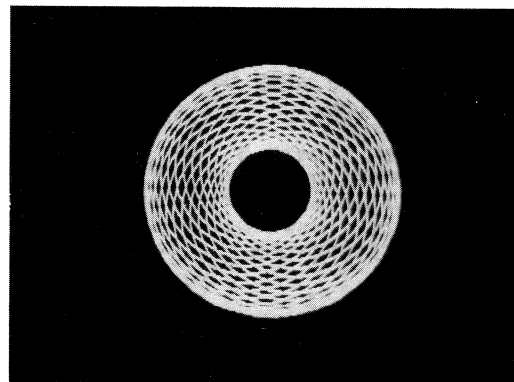
```

110 SCREEN 1:WIDTH 80
120 PAI=3.141593
130 LR=90:SR=65:R=50
140 RR=LR-SR:PR=LR/SR
150 SX=2.5:GX=320
160 SY=-1:GY=100
170 ST=PAI/90
180 * (Start position)
190 GOSUB 270
200 LINE(X,Y)-(X,Y),PSET

220 FOR TH=0 TO 82 STEP ST
230 GOSUB 270
240 LINE-(X,Y),PSET
250 NEXT
260 END
270 * Graph sub routine
280 FAI=TH*PR
290 X=RR*COS(TH)-R*COS(TH+FAI)
300 Y=RR*SIN(TH)-R*SIN(TH+FAI)

320 X=X*SX+GX
330 Y=Y*SY+GY
340 RETURN

```



PROGRAM 16; Neutral Tint (Color)

```

110 WIDTH40:COLOR 7,0:SCREEN 0,,0
120 LOCATE 15,1:PRINT"COLOR "
130 FOR I=0 TO 7
140 READ A$
150 X=(I MOD 4)*10+2
160 Y=(I*4)*10+12
170 LOCATE X,Y:PRINTA$;
180 NEXT

200 FOR I=8 TO 15
210 XO=(I MOD 4)*10+2
220 X1=XO+5
230 YO=(I*4-2)*10+5
240 Y1=YO+5
250 LINE(XO,YO)-(X1,Y1)," ",I,BF
260 LINE(XO,YO)-(X1,Y1)," ",I,B
270 NEXT

290 I=1:LOCATE 0,0,1
300 IF INPUT$(1)!="E" THEN 330
310 COLOR ,I:I=I+1:IF I>7 THEN I=0
320 GOTO 300
330 LOCATE 0,0,3:END
340 DATA "BLK","BLU","RED","MAG"
350 DATA "GRN","CYAN","YELLOW","WHITE"

```

```

110 COLOR 7,0:SCREEN 1:WIDTH 80
120 FOR I=0 TO 7
130 FOR J=0 TO 7
140 X=I*80:Y=J*24
150 FOR K=0 TO 11
160 V=K*2+Y
170 LINE(X,V)-(X+79,V),PSET,I
180 LINE(X,V+1)-(X+79,V+1),PSET,J
190 NEXT K,J,I
200 I$=INPUT$(1)

```

(Hints to make the delimit
beautiful)

```

LINE(0,100)-(100,150),PSET,4,B
PAINT(1,101),1,4

```

```

LINE(111,100)-(217,150),PSET,4,B
PAINT(112,101),2,4

```

```

LINE(0,150)-(150,150),PSET,4
LINE(75,120)-(75,180),PSET,1

```

```

LINE(245,120)-(245,180),PSET,2
LINE(170,150)-(320,150),PSET,4

```

PROGRAM 17; Move the graphics

```

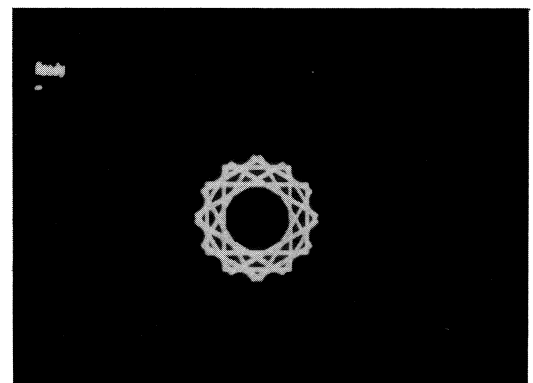
110 SCREEN 1:WIDTH80
120 DIM A(3,1):TH=3.141593/6
130 FOR I=0 TO 3
140 READ A(I,0),A(I,1)
150 NEXT
160 FOR K=0 TO 6
170 CLS:GOSUB 220
180 GOSUB 320
190 NEXT
200 DATA 30,20,-30,20,-30,-20,30,-20
210 END

230 J=3:GOSUB 290
240 LINE(XO,YO)-(XO,YO),PSET
250 FOR J=0 TO 3
260 GOSUB 290
270 LINE-(XO,YO),PSET
280 NEXT:RETURN

300 XO=A(J,0)*2.5+320:YO=100-A(J,1)
310 RETURN

330 FOR I=0 TO 3
340 X=A(I,0):Y=A(I,1)
350 A(I,0)=X*COS(TH)-Y*SIN(TH)
360 A(I,1)=X*SIN(TH)+Y*COS(TH)
370 NEXT:RETURN

```



PROGRAM 18; Enlarge/Compress the graphics;

```

110 SCREEN 1:WIDTH80
120 DIM A(3,1):A=.75
130 FOR I=0 TO 3
140 READ A(I,0),A(I,1)
150 NEXT I
155 FOR A=.75 TO 1.35 STEP .6
160 FOR K=1 TO 10
170 GOSUB 220
180 GOSUB 320
190 NEXT K,A
200 DATA 120,80,-120,80,-120,-80,120,-80

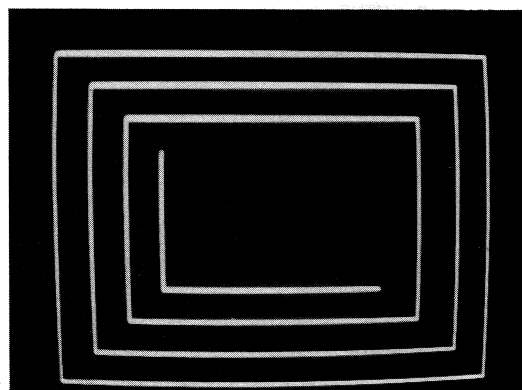
210 END

230 CLS:J=3:GOSUB 290
240 LINE(X0,Y0)-(X0,Y0),PSET
250 FOR J=0 TO 3
260 GOSUB 290
270 LINE-(X0,Y0),PSET
280 NEXT J:RETURN

300 X0=A(J,0)*2.5+320:Y0=100-A(J,1)
310 RETURN

330 FOR I=0 TO 3
340 A(I,0)=A*A(I,0)
360 A(I,1)=A*A(I,1)
370 NEXT I:RETURN

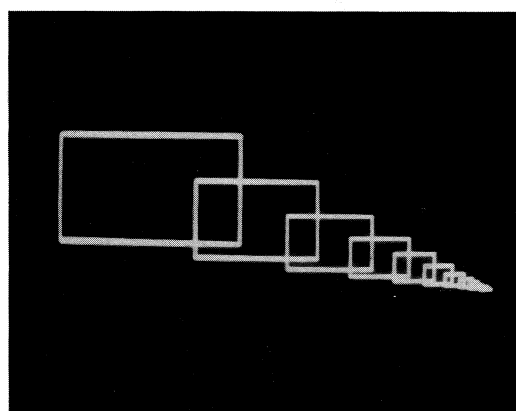
```



```

110 AS=TIMES:A=0
120 FOR I=1 TO 3
130 A=A+VAL(AS)
140 AS=MID$(AS,INSTR(AS,"")+1)
150 NEXT I
160 RANDOMIZE A
170 SCREEN 1:WIDTH80
180 DIM A(2,1):A=2/3
190 FOR J=0 TO 2
200 A(J,0)=639*RND
210 A(J,1)=199*RND
220 NEXT J:CLS
230 CC=6.5*RND+1
240 FOR J=0 TO 10
250 LINE(A(0,0),A(0,1))-(A(1,0),A(1,1)),
PSET,CC,B
260 A(0,0)=A*(A(0,0)-A(2,0))+A(2,0)
270 A(0,1)=A*(A(0,1)-A(2,1))+A(2,1)
280 A(1,0)=A*(A(1,0)-A(2,0))+A(2,0)
290 A(1,1)=A*(A(1,1)-A(2,1))+A(2,1)
300 NEXT J:GOTO 190

```



PROGRAM 19; DRAW statement

```

100 *****
105 * *
110 * DRAW PROGRAM *
115 * *
120 * *
125 * *
130 *****
135 ON ERROR GOTO 165:GOSUB 670
140 LINEINPUT">";MOVE$
145 GOSUB 200:GOTO 140
150 * END
155 CONSOLE 0,25
160 ON ERROR GOTO 0:SCREEN,,1:END
165 * ERROR
170 A=INSTR(MOVE$,DRAW$)
175 IF A=0 THEN 195
180 PRINTSPC(A)"*";:BEEP
185 IF INKEY$<>" " THEN 185
190 A$=INPUT$(1):PRINT
195 RESUME 140
200 *** DRAW ***
205 DRAW$=MOVE$
210 IF DRAW$="" THEN RETURN
215 A$=LEFT$(DRAW$,1)
220 DRAW$=MID$(DRAW$,2)
225 A=INSTR(CM$,A$)
230 IF A=0 THEN ERROR 5
235 ON A GOSUB 500,345,380,150,325,395,4
20,335,360,405,480,370,250,530
240 IF DRAW$<>" " THEN 215
245 RETURN
250 * EXECUTE <Xn(string)>
255 GOSUB 610:IF L=0 THEN L=1
260 IF LEFT$(DRAW$,1)<>"(" THEN ERROR 5
265 DR$=MID$(DRAW$,2)
270 FOR CNT=1 TO L:DRAW$=DR$
275 A$=LEFT$(DRAW$,1):A=INSTR(CM$,A$)
280 DRAW$=MID$(DRAW$,2)
285 IF A=0 THEN ERROR 5
290 ON A GOSUB 500,345,380,150,325,395,4
20,335,360,405,480,370,320,530
295 IF DRAW$="" THEN ERROR 5
300 IF ASC(DRAW$)<>ASC("(") THEN 275
305 NEXT
310 DRAW$=MID$(DR$,INSTR(DR$,"")+1)
315 RETURN
320 ERROR 5
325 * PEN UP <F>
330 P=0:RETURN
335 * PEN DOWN <P>
340 P=1:LINE(X,Y)-(X,Y),PSET,C:RETURN
345 * COLOR <Cn>
350 GOSUB 610:IF M=0 THEN C=0
355 C=(C+L) MOD 8:RETURN
360 * CLEAR <Q[n]>
365 GOSUB 610:COLOR,L:CLS:RETURN
370 * MOVE UP <Uy>
375 GOSUB 610:DY=-L:GOTO 390
380 * MOVE DOWN <Dy>
385 GOSUB 610:DY=L
390 DX=0:GOTO 450
395 *MOVE LEFT <Lx>
400 GOSUB 610:DX=-L:GOTO 415
405 * MOVE RIGHT <Rx>
410 GOSUB 610:DX=L
415 DY=0:GOTO 450
420 * MOVE <M[±]x,[±]y>
425 GOSUB 585:IF DM OR M THEN 440
430 * ABSOLUTE MOVE
435 X=DX:Y=L/HY:GOTO 465
440 * RELATIVE MOVE
445 DY=L

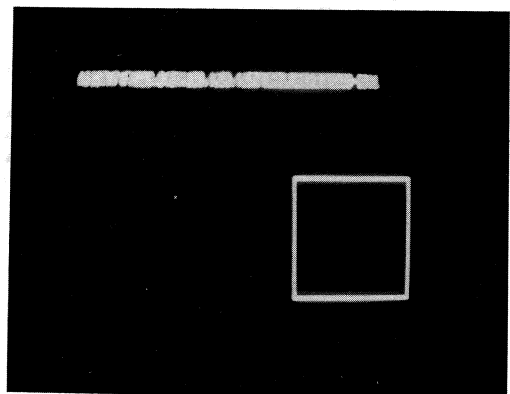
```

```

450 * ANGLE CAL.
455 DX=DX*S:DY=DY*S
460 X=FNAX(DX,DY):Y=FNAY(DX,DY)
465 * PLOT
470 IF P THEN LINE-(X,Y),PSET,C
475 RETURN
480 * SCALE <S[±]n>
485 GOSUB 610:IF M=0 THEN S=0
490 S=S*L/4:IF S<=0 THEN S=4
495 RETURN
500 * ANGLE <A[±]n>
505 GOSUB 610:IF M=0 THEN ANG=0
510 ANG=ANG+L*PAI/180
515 AA=COS(-ANG):AB=SIN(ANG)
520 AC=SIN(-ANG):AD=COS(ANG)
525 RETURN
530 * PAINT <Z[±]x,[±]y[,c]>
535 GOSUB 585:IF DM OR M THEN 550
540 * PAINT(ABSOLUTE)
545 SX=DX:SY=L/HY:GOTO 565
550 * PAINT(RELATIVE)
555 DX=DX*S:DY=L*S
560 SX=FNAX(DX,DY):SY=FNAY(DX,DY)
565 * EXECUTE PAINT
570 IF ASC(DRAW$)<>44 THEN SC=C:GOTO 580
575 GOSUB 605:SC=L
580 PAINT(SX,SY),SC:RETURN
585 * GET 2 NUMBER
590 GOSUB 610:DX=L:DM=M
595 IF DRAW$="" THEN ERROR 5
600 IF ASC(DRAW$)<>44 THEN ERROR 5
605 DRAW$=MID$(DRAW$,2)
610 * GET NUMBER
615 M=0:A$=""
620 IF DRAW$="" THEN L=0:RETURN
625 L$=LEFT$(DRAW$,1)
630 IF L$="+" THEN M=1:GOTO 650
635 IF L$="-" THEN M=1:GOTO 650
640 IF L$="." THEN 650
645 IF L$<"0" OR L$>"9" GOTO 660
650 A$=A$+L$:DRAW$=MID$(DRAW$,2)
655 IF DRAW$<>" " GOTO 625
660 L=VAL(A$)
665 RETURN
670 * INITIALIZE
675 CONSOLE 0,2,0
680 COLOR 7,0:SCREEN 1,,0:WIDTH 80
685 DRAW$="":L=0:X=0:Y=0:P=1:C=7:S=1
690 AA=1:AB=0:AC=0:AD=1:ANG=0:HY=2.4
695 PAI=3.141593:CM$="ACDEFLMPORSUXZ"
700 DEF FNAX(DX,DY)=AA*DX+AB*DY+X
705 DEF FNAY(DX,DY)=(AC*DX+AD*DY)/HY+Y
710 RETURN

```

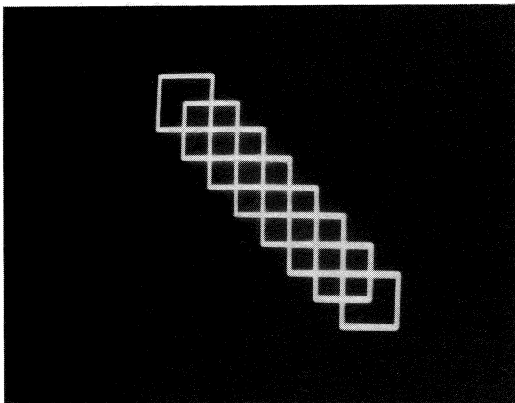
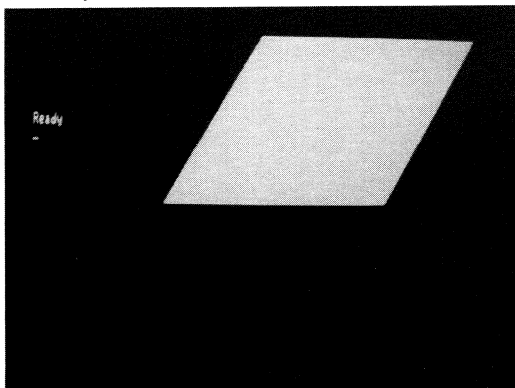
>C4AS4FM240,120PR160D160L160U160



Try this program!

>AS1C7X7(FM320,240M+100,+100PU200L2
00D200R200C-1S+1)

```
>AS4C7FM280,120P
>R200M-40,+160L200M+40,-160
>Z+10,+10,5
```



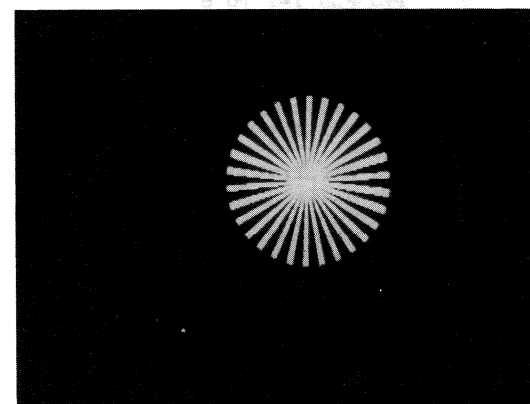
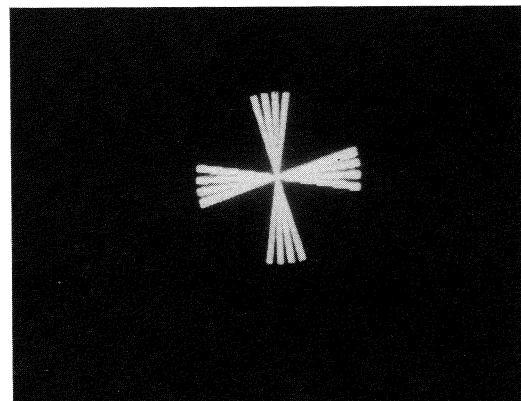
PROGRAM 20; Time tunnel

```
110 COLOR 7,0:SCREEN 1,,0:WIDTH80
120 DIM A(3),B(3,31)
130 FOR I=0 TO 3:READ A(I):NEXT
140 ZO=4200:Z=200
150 SX=3*40*1000:DX=320
160 SY=40*1000:DY=100
170 ' Main program
180 FOR I=0 TO 3
190 LINE(B(I,0),B(I,2))-(B(I,1),B(I,3)),
PRESET,B
200 B(I,0)=SX/A(I)+DX
210 B(I,1)=-SX/A(I)+DX
220 B(I,2)=SY/A(I)+DY
230 B(I,3)=-SY/A(I)+DY
240 LINE(B(I,0),B(I,2))-(B(I,1),B(I,3)),
PSET,I+1,B
250 A(I)=A(I)-Z
260 IF A(I)<=Z THEN A(I)=ZO
270 NEXT
280 GOTO 180
290 DATA 4000,3000,2000,1000
```

```
>AS4C0X36 (C+1FM320,240PU100A+10)
```

```
>AS4C5FM480,240PX72 (U10A+5) .
```

```
>C1AS6FM0,40P
>X7 (PR50D50L50U50FM+25,+25C+1)
```



PROGRAM 21; Rotation of the cubic

```

110 COLOR 7,0:SCREEN 1,,0:WIDTH40
120 DIM A(8,3),B(8,2)
130 FOR I=1 TO 8:FOR J=1 TO 3
140 READ A(I,J)
150 NEXT J,I
160 S=SIN(3.141593/18)
170 C=COS(3.141593/18)
180 FOR TH=1 TO 36
190 FOR I=1 TO 8
200 X=A(I,1):Z=A(I,3)
210 A(I,1)=C*X-S*Z
220 A(I,3)=S*X+C*Z
230 Y=A(I,2):Z=A(I,3)
240 A(I,2)=C*Y-S*Z
250 A(I,3)=S*Y+C*Z
260 B(I,1)=960*A(I,1)/(A(I,3)+480)+320
270 B(I,2)=400*A(I,2)/(A(I,3)+480)+100
280 NEXT
290 CLS
300 LINE(B(1,1),B(1,2))-(B(2,1),B(2,2)),
PSET,4

```

```

310 LINE-(B(3,1),B(3,2)),PSET,4
320 LINE-(B(4,1),B(4,2)),PSET,4
330 LINE-(B(1,1),B(1,2)),PSET,4
340 LINE-(B(5,1),B(5,2)),PSET,4
350 LINE-(B(6,1),B(6,2)),PSET,4
360 LINE-(B(7,1),B(7,2)),PSET,4
370 LINE-(B(8,1),B(8,2)),PSET,4
380 LINE-(B(5,1),B(5,2)),PSET,4
390 LINE(B(2,1),B(2,2))-(B(6,1),B(6,2)),
PSET,4
400 LINE(B(3,1),B(3,2))-(B(7,1),B(7,2)),
PSET,4
410 LINE(B(4,1),B(4,2))-(B(8,1),B(8,2)),
PSET,4
420 NEXT
430 DATA 40,40,40,-40,40,40,-40,40,-40
440 DATA 40,40,-40,40,-40,40,-40,-40
450 DATA 40,-40,-40,-40,40,-40,-40
460 END

```

PROGRAM 22; Color graphics Demo

```

10 '*****
20 '*'
30 '*'
40 '*'
50 '*' SAMPLE (GRAPHDEMO)
60 '*'
70 '*' 15,FEB,1980
80 '*'
90 '*****
100 '
110 PI=3.14159
120 SCREEN 1,1,1:CONSOLE0,25,0
130 CLS
140 DEFFNX(R)=FIX(COS(R)^3*I*5)+320
150 DEFFNY(R)=FIX(SIN(R)^3*I*5)+150
160 FOR I=1 TO 100
170 X1=RND*639:Y1=RND*199
180 IF Y1>150 AND X1>250 AND X1<450
THEN 170
190 PSET(X1,Y1,INT(RND*7+1)):NEXT
200 C=1:E=PI:S=.05
210 FOR I=13 TO 9 STEP -1
220 IF I=9 THEN E=PI*2
230 FOR R=0 TO E STEP S
240 X=FIX(COS(R)^3*I*4*5)+320
250 Y=FIX(SIN(R)^3*I*3)+I*15-100
260 LINE(X,Y)-(320,I*15),PSET,C
270 NEXT: C=C+1: IF C>7 THEN C=1
280 NEXT

```

PROGRAM 23; Screen reverse program

Input

CLEAR 300,&H6FFF

EXEC &H7000

```

00100 *****
00110 *
00120 * SCREEN REVERSE PROGRAM *
00130 *
00140 *
00150 *
00160 *****
7000 00170 ORG $7000
00180 *
00190 REVRSE LDB CSR
7000 F6 FFDB 00200 PSHS B
7003 34 04 00210 LDA #0
7005 B6 00 00220 STA CSR
700A BE 00A4 00230 LDX VTOP
700D A6 00 00240 LOOP LDA 0,X
700F F6 FFDB 00250 LDB CSR
7012 C4 1F 00260 ANDB #$1F
7014 C8 08 00270 EORB #$08
7016 F7 FFDB 00280 STB CSR
7019 A7 80 00290 STA ,X+
701B BC 00A5 00300 CMPX VEND
701E 26 ED 00310 BNE LOOP
7020 35 04 00320 PULS B
7022 C4 1F 00330 ANDB #$1F
7024 F7 FFDB 00340 STB CSR
7027 39 00350 RTS
00360 *
00A4 00370 VTOP EQU $00A4
00A5 00380 VEND EQU $00A5
FFDB 00390 CSR EQU $0FFDB
00400 *
0000 00410 END

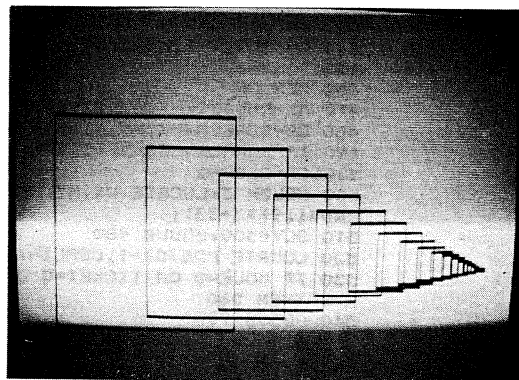
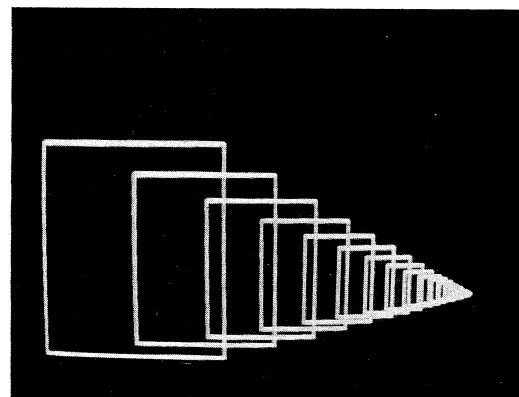
CSR FFDB F
LOOP 700D
REVRSE 7000
VEND 00A5 F
VTOP 00A4 F

```

*D7000

```

7000 F6 FF DB 34 04 B6 00 B7
7008 FF DB BE 00 A4 A6 00 F6
7010 FF DB C4 1F C8 08 F7 FF
7018 DB A7 80 BC 00 A5 26 ED
7020 35 04 C4 1F F7 FF DB 39
7028 00 00 00 00 00 00 00 00
7030 00 00 00 00 00 00 00 00
7038 00 00 00 00 00 00 00 00
*
```



PROGRAM 24; Light pen interrupt. Mole hunt game

```

100 CONSOLE 0,25,0:WIDTH 40
110 FOR I=0 TO 7
120 X1=I*10 MOD 40:X2=X1+9
130 Y1=(I*4)*12+1:Y2=Y1+12
140 CL=(I MOD 7)+9
150 LINE(X1,Y1)-(X2,Y2)," ",CL,BF
160 IF I THEN PEN I:(X1,Y1)-(X2,Y2)
170 LOCATE X1+2,Y1+5:COLOR CL
180 PRINT "AREA" I
190 NEXT I

```

```

200 ON PEN GOSUB 300,310,320,330,340,350
,360,370
210 PEN ON:COLOR 7,0
220 GOTO 220
300 LOCATE 0,0:PRINT "No area":RETURN
310 LOCATE 0,0:PRINT "Area 1 ":RETURN
320 LOCATE 0,0:PRINT "Area 2 ":RETURN
330 LOCATE 0,0:PRINT "Area 3 ":RETURN
340 LOCATE 0,0:PRINT "Area 4 ":RETURN
350 LOCATE 0,0:PRINT "Area 5 ":RETURN
360 LOCATE 0,0:PRINT "Area 6 ":RETURN
370 LOCATE 0,0:PRINT "Area 7 ":RETURN

```

```

100 '
110 ' mole hunt game
120 '
130 ' starting message
140 SCREEN 1,,0:CONSOLE 0,25,0:WIDTH 40:
COLOR 1,5:CLS
150 Z$="Level-3 MOLE HUNTING"
160 LOCATE 7,12:GOSUB 600
170 DLY=300:GOSUB 980
180 RANDOMIZE (PEEK(&HFF0)*255+PEEK(&HFF
E0))
190 CLS:COLOR 3
200 LOCATE 3,2:PRINT "3 TICKETS"
210 LOCATE 3,4:PRINT "4 POINTS";
220 LOCATE POS(0)-7,CSRLIN+1:PRINT" # is
2 POINTS"
230 LOCATE 3,7:COLOR 2:PRINT" BREAK
THS LAW"
240 LINE(23,80)-(72,105),PSET,4,BF
250 LINE(23,120)-(72,143),PSET,2,B:PAINT
(36,132),4,2
260 COLOR 1:LOCATE 5,11:PRINT": HIT"
270 LOCATE 5,16:PRINT": SANCTUARY"
280 LOCATE 3,19:COLOR 2:PRINT" GAME
END IF NO MOLES OR NO TICKETS"
290 DLY=2000:GOSUB 980
300 ' set up
310 COLOR 12,4:CLS:LINE(0,0)-(639,199),"
",12,BF
320 TM=100
330 LINE(400,80)-(639,199),PSET,2,B
340 PAINT (416,81),4,2
350 MOLE=30:CLOP=20:TICKET=3:SCORE=0
360 PEN OFF:PEN 1:(25,10)-(39,24)
370 COLOR 6,4
380 FOR I=1 TO CLOP
390 CX=INT(RND*39):CY=INT(RND*23.9+1)
400 IF CX>24 AND CY>5 THEN 390
410 LOCATE CX,CY:PRINT"X":NEXT I
420 GOSUB 700
430 ON PEN GOSUB 810,910
440 C=ASC("X"):MM=ASC("&"):MF=ASC("#")
450 ' game
460 PEN ON
470 MX=RND*39:MY=RND*23.9+1
480 CP=SCREEN(MX,MY,1) MOD 16
490 IF SCREEN(MX,MY)=C THEN SCORE=SCORE-
1:CLOP=CLOP-1
500 COLOR 3:LOCATE MX,MY:PRINT CHR$(INT(
RND*1.9)*3+35)
510 DLY=300:GOSUB 980
520 LOCATE POS(0)-1,CSRLIN:PRINT" ";
530 IF MOLE=0 OR TICKET=0 OR CLOP=0 OR T
M=0 THEN 560
540 GOSUB 700

```

```

550 GOTO 470
560 LINE(10,10)-(30,12)," ",10,BF
570 LOCATE 15,11:COLOR 10:Z$="GAME OVER"
:GOSUB 600:LOCATE 0,0,1
580 S$=INPUT$(1):RUN
600 ' starting message
610 FOR IZ=1 TO LEN(Z$)
620 PRINT USING"!";MID$(Z$,IZ,1);
630 DLY=100:GOSUB 980
640 NEXT IZ
650 RETURN
700 ' score display
710 LOCATE 1,0,3:COLOR 1
720 PRINT USING"POINT=*****";SCORE;:PRINT
USING"MOLES=***";MOLE;:PRINT USING"Ticket
=***";TICKET;:PRINT USING"TIME=***";TM
730 RETURN
800 ' hit field
810 PEN OFF
820 P=PEN(0):PX=PEN(1):PY=PEN(2)
830 PS=SCREEN(PX,PY)
840 IF PS=MM THEN SCORE=SCORE+4:MOLE=MOLE
-1:PEN ON:RETURN
850 IF PS=MF THEN SCORE=SCORE+2:MOLE=MOLE
-1:PEN ON:RETURN
860 TM=TM-1:PEN ON:RETURN
900 ' hit area
910 PEN OFF
920 P=PEN(0):PX=PEN(1):PY=PEN(2)
930 PS=SCREEN(PX,PY)
940 IF PS=MM OR PS=MF THEN SCORE=SCORE-5
:MOLE=MOLE-1:TICKET=TICKET-1
970 TM=TM-1:PEN ON:RETURN
980 ' delay
990 FOR DI=0 TO DLY:NEXT DI
1000 RETURN

```

PROGRAM 25; Hits for RS232C port

```

100 SCREEN 0,,0:WIDTH 80:CONSOLE 1,24,0
110 ON COM(1) GOSUB 210
120 OPEN "I",#1,"COM1:(F8N1)"
130 LINEINPUT "INPUT FILE NAME ? : ";F$
140 FLN$=LEFT$(F$,8)
150 OPEN "O",#2,FLN$
160 TIME$="00:00:00":T=0:L=1:C=0
170 COM(1) ON:CLS
180 LOCATE 0,0:PRINT TIME$
190 IF INKEY$=CHR$(27) THEN 200 ELSE 180
200 CLS:PRINT "Working time" T"sec.":END
210 T=TIME:LOCATE C,L+1
220 LINEINPUT #1,A$
230 PRINT:PRINT #2,A$
240 PRINT A$
250 L=CSRLIN:C=POS(0)
260 T=T+(TIME-T0)
270 RETURN

```

PROGRAM 26; Error test

```

100 SCREEN 0,,0:WIDTH40:CONSOLE 0,23,0
110 ON ERROR GOTO 200
120 RANDOMIZE
130 CL=RND*6+1:ML=RND*50
140 FOR I=0 TO ML
150 READ Z$
160 NEXT I
170 IF INKEY$="" THEN ELSE 170
180 COLOR CL:PRINT:PRINT Z$
190 GOTO 130
200 IF ERR=4 AND ERL=150 THEN ELSE ON ER
ROR GOTO 0
210 RESTORE
220 RESUME NEXT
300 DATA "It's no use crying over spilt
milk."
310 DATA "Time flies like an arrow."
320 DATA "In Rome,do as Roman's do."
330 DATA "Tomorrow never comes."
340 DATA "A drowning man will catch at a
straw."
350 DATA "History repeats itself."
360 DATA "Set a thief to catch a thief."
370 DATA "That which is bought cheap is
the dearest."
380 DATA "Blood is thicker than water."
390 DATA "East,west,home is best."

```

```

100 ON ERROR GOTO 10000
110 SCREEN 1:WIDTH 80:CONSOLE 24,1,0
120 ERROR 104
130 INPUT "DRAW Y=X GRAPH (Y or N) "
;YN$
140 IF YN$="Y" THEN ELSE 170
150 FOR X=-5 TO 5 STEP .1:CL=6:Y=X:ERROR
103:NEXT X
160 GOSUB 500
170 INPUT "DRAW Y=X^2 GRAPH (Y or N)
";YN$
180 IF YN$="Y" THEN EL 200
190 CL=5
200 FOR I=0 TO 3.3 STE 1

```

```

210 X=I
220 Y=X^2:ERROR 105
230 X=-I:ERROR 105
240 NEXT I
250 GOSUB 500
260 INPUT "DRAW Y=X^3 GRAPH (Y or N) "
;YN$
270 IF YN$="Y" THEN ELSE 320
280 CL=4
290 FOR X=-2.3 TO 2.2 STEP .05
300 Y=X^3:ERROR 105
310 NEXT X
320 GOSUB 500
330 GOTO 130
500 INPUT "ERASE A PART (Y or N) "
;YN$
510 IF YN$="Y" THEN ELSE RETURN
520 INPUT "WHICH PART (1 - 4) ";A
REA
530 IF AREA<1 OR AREA>4 THEN 520
540 ERROR AREA+99
550 ERROR 104
560 RETURN
10000 IF ERR<100 THEN ON ERROR GOTO 0
10010 ER=ERR-99
10020 ON ER GOSUB 10100,10200,10300,1040
0,10500,10600
10030 RESUME NEXT
10100 ' Clear screen area 1
10110 LINE(40,0)-(79,11)," ",7,BF
10120 RETURN
10200 ' Clear screen area 2
10210 LINE(0,0)-(39,11)," ",7,BF
10220 RETURN
10300 ' Clear screen area 3
10310 LINE(0,12)-(39,24)," ",7,BF
10320 RETURN
10400 ' Clear screen area 4
10410 LINE(40,12)-(79,24)," ",7,BF
10420 RETURN
10500 ' Draw x=0,y=0
10510 LINE(0,93)-(639,93),PSET,1
10520 LINE(319,0)-(319,199),PSET,1
10530 RETURN
10600 PSET(X*60+319,93-Y*8,CL)
10610 RETURN

```


PROGRAM 27; Write the machine code

```

100 '*** WRITE CODE IN MEMORY ***
110 CLEAR 300,&H6FFF
120 ADR=&H7000
130 FOR I=0 TO 39
140 READ A$
150 POKE ADR+I,VAL("&H"+A$)
160 NEXT
170 '*** DATA ***
180 '%% Reverse %%
190 DATA F6,FF,DB,34,04,B6,00,B7
200 DATA FF,DB,BE,00,A4,A6,00,F6
210 DATA FF,DB,C4,1F,C8,08,F7,FF
220 DATA DB,A7,B0,BC,00,A5,26,ED
230 DATA 35,04,C4,1F,F7,FF,DB,39

```

PROGRAM 28; Memory usage information

```

100 '*** Used Memory Information ***
110 DEF FNA(X)=PEEK(X)*256+PEEK(X+1)
120 PS=0:PE=0:VE=0:AE=0
130 PS=FNA(&H1D):PE=FNA(&H1F)
140 VE=FNA(&H21):AE=FNA(&H23)
150 PRINT USING"##### bytes used for pro
gram";PE-PS
160 PRINT USING"##### bytes used for sim
ple variable";VE-PE
170 PRINT USING"##### bytes used for arr
ay";AE-VE
180 END

```

Ready
go.100

```

293 bytes used for program
40 bytes used for simple variable
0 bytes used for array

```

Ready

PROGRAM 29 ; Dump list program

Input

CLEAR 300,&H70FF

```

100 '*** DUMP PROGRAM ***
110 DEFUSR=&H7102
120 ADR=&H7100
130 I%=USR(-1%)
140 A$=USR("")
150 IF A$="" THEN END
160 I%=PEEK(ADR)
170 IF I%=2 THEN PRINT A$%="USR(0%)
180 IF I%=3 THEN PRINT A$%="USR(1%)
190 IF I%=4 THEN PRINT A$%="USR(2%)
200 IF I%=8 THEN PRINT A$%="USR(0#)
210 GOTO 140

```

GOTO100
 ADR!= 28928
 I%= 2
 A\$=A

	00100 *		
	00110 * DUMP SUB ROUTINE		
	00120 *		
	00130 * BY M.MORI (ESC)		
	00140 *		
7100	00150	ORG	\$7100
	00160 *		
7100	00170	MODE	RMB 1
7101	00180	FLAG	RMB 1
	00190 *		
7102 B1 02	00200	DUMP	CMPA #2
7104 26 11	00210	BNE	DUMP1
7106 10AE 02	00220	LDY	2,X
7109 27 0C	00230	BEQ	DUMP1
710B 7F 7101	00240	CLR	FLAG
710E 10BE 001F	00250	LDY	VALTOP
7112 10BF 7176	00260	STY	TOP
7116 39	00270	RTS1	RTS
	00280 *		
7117 73 7101	00290	DUMP1	COM FLAG
711A F6 7101	00300	LDB	FLAG
711D 27 2B	00310	BEQ	DUMP4
711F 81 03	00320	CMPA	#3
7121 27 03	00330	BEQ	DUMP2
7123 7E AC06	00340	ERR1	JMP ERRORS
	00350 *		
7126 10BE 7176	00360	DUMP2	LDY TOP
712A 10BC 0021	00370	CMFY	VALEND
712E 27 E6	00380	BEQ	RTS1
7130 A6 A4	00390	LDA	,Y
7132 E6 A0	00400	LDB	,Y+
7134 C4 0F	00410	ANDB	#\$0F
7136 5C	00420	INCB	
7137 E7 84	00430	STB	,X
7139 10AF 01	00440	STY	1,X
713C 31 A5	00450	LEAY	B,Y
713E 10BF 7176	00460	STY	TOP
7142 44	00470	LSRA	
7143 44	00480	LSRA	
7144 44	00490	LSRA	
7145 44	00500	LSRA	
7146 B7 7100	00510	STA	MODE
7149 39	00520	RTS	
	00530		
714A 10BE 7176	00540	DUMP4	LDY TOP
714E B1 7100	00550	CMPA	MODE
7151 26 D0	00560	BNE	ERR1
7153 34 12	00570	PSHS	X,A

```

7155 81 02      00580      CMPA   #2
7157 26 02      00590      BNE    DUMP5
7159 30 02      00600      LEAX   2,X
715B E6 A0      00610 DUMP5 LDB    ,Y+
715D E7 80      00620      STB    ,X+
715F 4A         00630      DECA
7160 26 F9      00640      BNE    DUMP5
7162 10BF 7176  00650      STY    TOP
7166 35 14      00660      PULS   B,X
7168 C1 03      00670      CMPB   #3
716A 23 09      00680      BLS    DUMP7
716C E6 01      00690      LDB    1,X
716E 10         00700      SEX
716F CA 80      00710      ORB    #80
7171 E7 01      00720      STB    1,X
7173 A7 08      00730      STA    B,X
7175 39         00740 DUMP7 RTS
              00750 *
7176          00760 TOP    RMB    2
              00770 *
              00780 VALTOP EQU    $1F
              0021    00790 VALEND EQU    $21
              00800 *
              AC06    00810 ERRORS EQU    $0AC06
              00820 *
              0000    00830      END

```

```

DUMP      7102
DUMP1     7117   F
DUMP2     7126   F
DUMP4     714A   F
DUMP5     715B   F
DUMP7     7175   F
ERR1      7123
ERRORS    AC06   F
FLAG      7101
MODE      7100
RTS1      7116
TOP       7176   F
VALEND    0021   F
VALTOP    001F   F

```

*07100

```

7100 00 00 81 02 26 11 10 AE
7108 02 27 0C 7F 71 01 10 BE
7110 00 1F 10 BF 71 76 39 73
7118 71 01 F6 71 01 27 2B 81
7120 03 27 03 7E AC 06 10 BE
7128 71 76 10 BC 00 21 27 E6
7130 A6 A4 E6 A0 C4 0F 5C E7
7138 B4 10 AF 01 31 A5 10 BF

```

*D

```

7140 71 76 44 44 44 44 B7 71
7148 00 39 10 BE 71 76 B1 71
7150 00 26 D0 34 12 81 02 26
7158 02 30 02 E6 A0 E7 80 4A
7160 26 F9 10 BF 71 76 35 14
7168 C1 03 23 09 E6 01 1D CA
7170 80 E7 01 A7 08 39 00 00
7178 00 00 00 00 00 00 00

```

PROGRAM 27; Sound Demo

```

100 *** SOUND ***
110 DEFUSR=&H7202
120 FOR I=0 TO 7
130 READ TN
140 GOSUB160
150 NEXT:END
160 LN%=(32767/TN)*2
170 POKE &H7200,TN*256
180 POKE &H7201,TN MOD 256
190 LN%=USR(LN%)
200 RETURN
210 DATA 80,86,98,110,125,131,148,169

```

```

                                00100 *
                                00110 * SOUND PROGRAM
                                00120 *
                                00130 *
                                00140 *
7200                            00150      ORG      $7200
                                00160 *
7200                            00170 TONE      RMB      2
                                00180 *
7202 AE      02      00190 SOUND  LDX      2,X
7204 FC      7200    00200 LOOP1  LDD      TONE
7207 83      0001    00210 LOOP2  SUBD     #1
720A 26      FB      00220        BNE     LOOP2
720C 73      721A    00230        COM     ONP
720F 86      721A    00240        LDA     ONP
7212 B7      FFD3    00250        STA     MUSIC
7215 30      1F      00260        LEAX    -1,X
7217 26      EB      00270        BNE     LOOP1
7219 39      00280        RTS
                                00290 *
721A                            00300 ONP      RMB      1
                                00310 *
                                FFD3    00320 MUSIC  EQU     $0FFD3
                                0000    00330 *
                                00340      END

```

LOOP1 7204
 LOOP2 7207

```

MUSIC  FFD3  F
ONP     721A  F
SOUND   7202
TONE     7200

```

*D7200

```

7200 00 00 AE 02 FC 72 00 83
7208 00 01 26 FB 73 72 1A B6
7210 72 1A B7 FF D3 30 1F 26
7218 EB 39 00 00 00 00 00 00
7220 00 00 00 00 00 00 00 00
7228 00 00 00 00 00 00 00 00
7230 00 00 00 00 00 00 00 00
7238 00 00 00 00 00 00 00 00

```

*

PROGRAM 28; To find the reserved statements

100 OPEN"O",1,"SCRN:"
 110 ADR=&HA078:N=127:F=-1
 120 N=N+1:A\$=RIGHT\$(" "+HEX\$(N),4)+" "
 130 K=PEEK(ADR):ADR=ADR+1
 140 IF K=0 THEN PRINT#1:END
 150 A\$=A\$+CHR\$(K AND 127)
 160 IF K<128 THEN 130
 170 PRINT#1,A\$
 180 IF N=220 THEN N=&HFF7F:F=0
 190 F=NOT F:IF F THEN PRINT#1
 200 GOTO 120

80 END
 82 NEXT
 84 DIM
 86 LET
 88 RUN
 8A RESTORE
 8C REM
 8E STOP
 90 TRON
 92 SWAP
 94 DEFINT
 96 DEFDBL
 98 WAIT
 9A EDIT
 9C RESUME
 9E DELETE
 A0 WIDTH
 A2 MON
 A4 CLS
 A6 PSET
 A8 MOTOR
 AA SAVE
 AC MERGE
 AE OPEN
 B0 FILES
 B2 KEY
 B4 BEEP
 B6 LINE
 B8 POKE
 BA CONT
 BC CLEAR
 BE WHILE
 C0 NEW
 C2 TO
 C4 FN
 C6 USING
 C8 ERL
 CA OFF
 CC NOT
 CE +
 DO *
 D2 ^
 D4 OR
 81 FOR
 83 DATA
 85 READ
 87 GO
 89 IF
 8B RETURN
 8D '
 8F ELSE
 91 TROFF
 93 DEFSTR
 95 DEFSGN
 97 ON
 99 RENUM
 9B ERROR
 9D AUTO
 9F TERM
 A1 UNLIST
 A3 LOCATE
 A5 CONSOLE
 A7 PRESET
 A9 SKIPF
 AB LOAD
 AD EXEC
 AF CLOSE
 B1 COM
 B3 PAINT
 B5 COLOR
 B7 DEF
 B9 PRINT
 BB LIST
 BD RANDOMIZE
 BF WEND
 C1 TAB(
 C3 SUB
 C5 SPC(
 C7 USR
 C9 ERR
 CB THEN
 CD STEP
 CF -
 D1 /
 D3 AND
 D5 XOR

D6 EQV
 D8 MOD
 DA >
 DC <
 FF80 SGN
 FF82 ABS
 FF84 POS
 FF86 LOG
 FF88 COS
 FF8A TAN
 FF8C PEEK
 FF8E STR\$
 FF90 ASC
 FF92 CINT
 FF94 CDBL
 FF96 SPACE\$
 FF98 OCT\$
 FF9A EOF
 FF9C LEFT\$
 FF9E MID\$
 FFA0 SCREEN
 FFA2 STRING\$
 FFA4 INKEY\$
 FFA6 CSRLIN
 FFA8 TIME
 D7 IMP
 D9 *
 DB =
 FF81 INT
 FF83 FRE
 FF85 SQR
 FF87 EXP
 FF89 SIN
 FF8B ATN
 FF8D LEN
 FF8F VAL
 FF91 CHR\$
 FF93 CSNG
 FF95 FIX
 FF97 HEX\$
 FF99 LOF
 FF9B PEN
 FF9D RIGHT\$
 FF9F INSTR
 FFA1 VARPTR
 FFA3 RND
 FFA5 INPUT
 FFA7 POINT
 FFA9 DATE